UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

Autor/es

DANIEL PÉREZ GARCÍA

Director/es

JAVIER ESTEBAN VICUÑA MARTÍNEZ

Facultad

Escuela Técnica Superior de Ingeniería Industrial

Titulación

Grado en Ingeniería Electrónica Industrial y Automática

Departamento

INGENIERÍA ELÉCTRICA

Curso académico

2017-18

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL

TRABAJO DE FIN DE GRADO

TITULACIÓN: Grado en
Ingeniería Electrónica Industrial y Automática

CURSO: 2017/2018 **CONVOCATORIA: SEPTIEMBRE**

TÍTULO:

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

AUTOR: Daniel Pérez García

DIRECTOR/ES: Javier Esteban Vicuña Martínez

DEPARTAMENTO: Ingeniería Eléctrica

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

Autor: Daniel Pérez García

Director: D. Javier Esteban Vicuña Martínez

Septiembre 2018

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

1. ÍNDICE

Daniel Pérez García

Septiembre 2018

INDICE GENERAL

2. MEMORIA

3. ANEXOS

6. MEDICIONES

7. PRESUPUESTO

INDICE DE ILUSTRACIONES

2. MEMORIA

3. ANEXOS

INDICE DE TABLAS

2. MEMORIA

3. ANEXOS

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

2. MEMORIA

Daniel Pérez García

Septiembre 2018

ÍNDICE DE LA MEMORIA

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE TABLAS

2.1 PRESENTACIÓN

Este trabajo ha sido realizado por Daniel Pérez García, alumno de cuarto curso del Grado en Ingeniería Electrónica Industrial y Automática y se presenta al objeto de obtener reconocimiento académico para la obtención del título de Graduado en Ingeniería Electrónica Industrial por la Universidad de La Rioja.

2.2 OBJETO

2.2.1 General

Este proyecto tiene como objeto principal el diseño y desarrollo de un sistema distribuido a modo de prototipo que monitorice y registre el histórico de variables de un proceso y posibilite el control de la calidad mediante la captura de datos para su posterior análisis, aplicado al proceso de fermentación del vino en bodega.

Este sistema estará formado por una red de nodos distribuidos y será capaz de tomar y almacenar información procedente de los distintos nodos de la red, acerca de las distintas variables que influyen en el proceso. Estos datos serán relevantes para su posterior análisis e interpretación por el enólogo. A través de este análisis, se podrán crear estrategias o realizar modificaciones en futuras elaboraciones aportando de esta manera cierto valor añadido al producto final.

Para configuración del funcionamiento de la red de nodos se desarrollará una aplicación de usuario que permitirá establecer el número de nodos y la configuración de cada uno de ellos: asignación de entradas, frecuencia de muestreo, adaptación de valores, etc. Además, desde esta aplicación se podrá acceder a una base de datos para consultar la información almacenada, generar gráficos, etc.

2.2.2 Objetivos particulares

- Diseño de una aplicación software para la configuración del funcionamiento de la red de nodos y acceso a la base de datos para su almacenamiento y visualización.
- Interconexión de los nodos mediante una red de comunicaciones vía CAN bus.
- Desarrollo de un único firmware permitiendo trabajar a cada nodo como maestro o como esclavo.
- Ensayos y pruebas necesarios para la comprobación del funcionamiento del prototipo.
- Documentación del proyecto

2.3 ALCANCE

El alcance de este proyecto contempla las fases necesarias para cumplir con los objetivos establecidos para llevar a cabo el desarrollo del prototipo. Dentro de estas fases se han previsto:

- Desarrollo de una aplicación capaz de comunicarse vía serie con un nodo maestro para la configuración de la red de nodos, así como la conexión con una base de datos para el almacenamiento y representación de los mismos.

- Desarrollo de un único firmware capaz de gobernar la red de nodos y permitir trabajar a cada nodo como esclavo o maestro.

- Consolidación de la red de comunicaciones vía CAN bus.

- Estudio y creación de las tramas necesarias para la comunicación entre los nodos, entre el nodo maestro y la aplicación, y entre la aplicación y la base de datos.

Este proyecto abarca el desarrollo de un sistema prototipo a nivel de laboratorio, quedando fuera del alcance de este proyecto la implementación física del sistema en un proceso de fermentación real.

Cada bodega presenta distintos elementos y condiciones de trabajo en el proceso de fermentación del vino. Esto implica que la elección de los sensores está muy ligada a cada proceso en concreto. Por ello se considerara resuelta para cada variable a medir, la elección del sensor y el acondicionamiento necesario para cada caso particular, y se considerarán las variables como lecturas de voltaje en los pines de entrada de las placas.

2.4 ANTECEDENTES

En este apartado se van a introducir conceptos relacionados con el proyecto los cuales han servido para el estudio de las distintas alternativas.

2.4.1 Proceso de vinificación

A continuación, se van a describir brevemente las fases para la vinificación a partir de la uva que llega a la bodega. Posteriormente, se desarrollará más a fondo el periodo de fermentación y cómo influyen las distintas variables del proceso en las características y calidad final del vino.

Despalillado y estrujado

A la llegada de la uva a la bodega, se toma una muestra donde se mide el grado Brix (cantidad de azúcares), pH y la acidez total. Se realiza la pesada de la uva y se descarga en la tolva para proceder al despalillado.

En este proceso se separa el raspón del grano de uva. Para ello se emplean máquinas denominadas despalilladoras. Consisten en un tambor perforado dentro del cual gira un eje con paletas. Mediante el giro de estas paletas los granos de una pasan a través de las perforaciones del tambor dejando dentro los raspones.

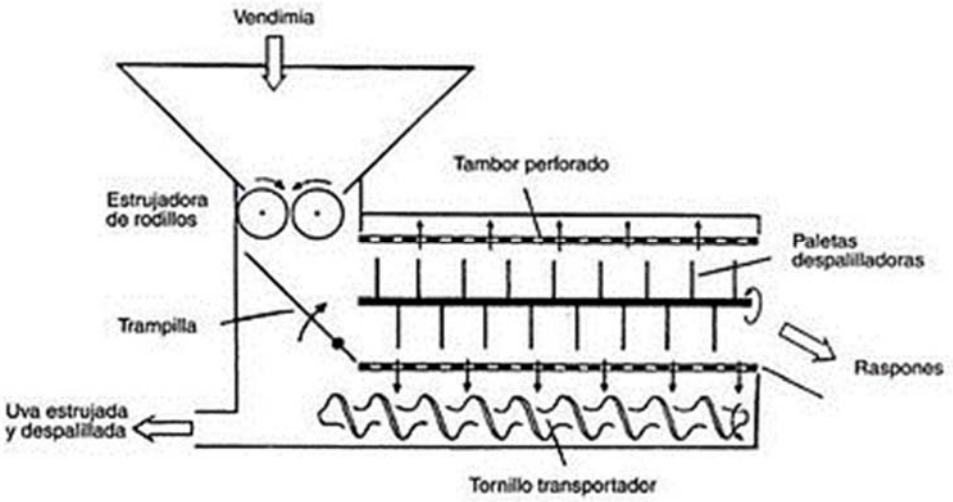


Ilustración 1- Esquema de una máquina despalilladora

En el estrujado (o molienda) se produce la rotura del hollejo provocando el desprendimiento de la pulpa para así extraer mejor el mosto. En algunos casos este proceso lo realizan las propias despalilladoras. Si no, se pasan los granos de uva por las denominadas trituradoras de rodillo. Este proceso no debe ser demasiado exhaustivo, ya que conviene evitar que se rompan las semillas, algo que podría aportar amargor al mosto.

Encubado y maceración

Una vez despalillada y estrujada la uva, se lleva hasta los depósitos para encubar e iniciar la maceración. Para ello se depositan en un mismo depósito el mosto y las partes sólidas, y se dejan macerar a temperatura controlada durante unos días. Se trata de un proceso determinante en la elaboración del vino ya que tanto los aromas como el color se encuentran en la parte de la piel. Los aromas y otras sustancias pasan desde los hollejos, semillas y raspones, al mosto. El objetivo de este proceso es la extracción óptima de estas sustancias: los denominados taninos buenos; y la extracción mínima de otras sustancias responsables de olores y sabores no deseables.

Remontados

Para favorecer esta extracción de aromas y sustancias acentuando la maceración, se lleva a cabo el denominado remontado. Consiste en extraer el mosto por la parte inferior del depósito y añadirlo por la parte superior. De esta manera se consigue una difusión de las sustancias del hollejo por toda la masa de líquido. También se logra distribuir las levaduras por toda la masa, las cuales son las protagonistas de la fermentación alcohólica.

Se suelen realizar dos remontados: uno por la tarde y otro por la mañana. Es importante que se lleven a cabo al principio de la fermentación o en las primeras horas, ya que de esta manera las levaduras podrán sacar más provecho al oxígeno que se les proporciona mediante esa aireación.

Fermentación alcohólica

En estos mismos depósitos donde se lleva a cabo la maceración, se lleva también a cabo el proceso de fermentación alcohólica. Esta es la etapa en la cual el azúcar de las uvas se transforma finalmente en alcohol etílico y gas carbónico (CO_2).

Dura en torno a 7-10 días. Durante el desarrollo de la fermentación, en los depósitos ocurre el fenómeno llamado estratificación en el cual los hollejos, pulpa y semillas se depositan o establecen en diferentes niveles del depósito.

Ilustración 2- Estratificación durante la fermentación

Se parte de una masa homogénea de mosto, hollejos, pulpa y semillas. Pasadas las horas ya se pueden diferenciar un estrato inferior de mosto y semillas, y otro superior de pulpas y

hollejos. En este punto, las levaduras encargadas de la fermentación actúan fuertemente en la parte superior, denominada sombrero.

Pasados un par de días, el estrato superior se va compactando y se inicia la denominada fermentación tumultuosa. Se denomina de esta manera ya que es la fase mas activa de la fermentación, durando entre 5 y 10 días. Las levaduras transforman el azúcar en alcohol, gas carbónico y calor, hasta tal punto que el mosto llega a bullir como si estuviera hirviendo.

A partir de los 8 días de fermentación, la parte superior ya está compactada y los sedimentos son mas abundantes. En este momento ya se puede proceder al descube de los depósitos.

Descube

El descubado o descube, consiste en extraer el líquido de la parte inferior del depósito para llevarlo a otro depósito si es que aún no la finalizado la fermentación. A este proceso se le llama 'sangrado'. El enólogo es el encargado de determinar el momento óptimo para llevarlo a cabo mediante el análisis de parámetros (densidad, alcohol, temperatura, etc). De esta decisión llevada a cabo por el enólogo depende el éxito o fracaso del proceso de vinificación.

Prensado

Una vez extraído el líquido del depósito, se recogen las pastas y se llevan a la prensa para aprovechar el vino que aún contienen. El vino procedente del sangrado es del de mayor calidad, por lo que se aprovecha para los mejores vinos. Conforme se llena la prensa se dejan escurrir las pastas de forma natural, obteniendo un vino de calidad similar a la del sangrado. Sin embargo, a medida que se van prensando las pastas se va obteniendo un vino de calidad inferior.

Ilustración 3 - Sección de una prensa de pastas

Con las pastas ya prensadas se realiza el trasiego, que consiste en separar las materias solidas depositadas en el fondo de los depósitos mediante el trasvase del vino a otros depósitos directamente a las barricas donde se llevará a cabo la fermentación maloláctica.

Fermentación maloláctica

Es la última etapa en el proceso de vinificación. En esta etapa se transforma el ácido málico en ácido láctico a través de bacterias que se encuentran de forma natural en el vino recién fermentado. Si este proceso se realiza correctamente, se evitarán problemas posteriores como fermentaciones indeseables ya que el ácido málico puede dar origen a la multiplicación de bacterias perjudiciales para la calidad del vino, como su aroma y su color. En definitiva, su principal propósito es el reducir la acidez del vino, por lo que se hace imprescindible para vinos con pH inferior a 3'5.

2.4.2 Parámetros a controlar en la etapa de fermentación

El objeto de este proyecto es desarrollar un prototipo capaz de tomar datos sobre distintos parámetros de este proceso permitiendo cierta trazabilidad del proceso. Además, mediante el análisis de esos datos y relacionándolos con el producto final, el enólogo podrá establecer estrategias o realizar modificaciones en futuras elaboraciones aportando de esta manera cierto valor añadido al producto final.

Las levaduras son las protagonistas de la fermentación alcohólica. Por ello es necesario saber qué factores influyen en el desarrollo de las levaduras y de qué modo lo hacen.

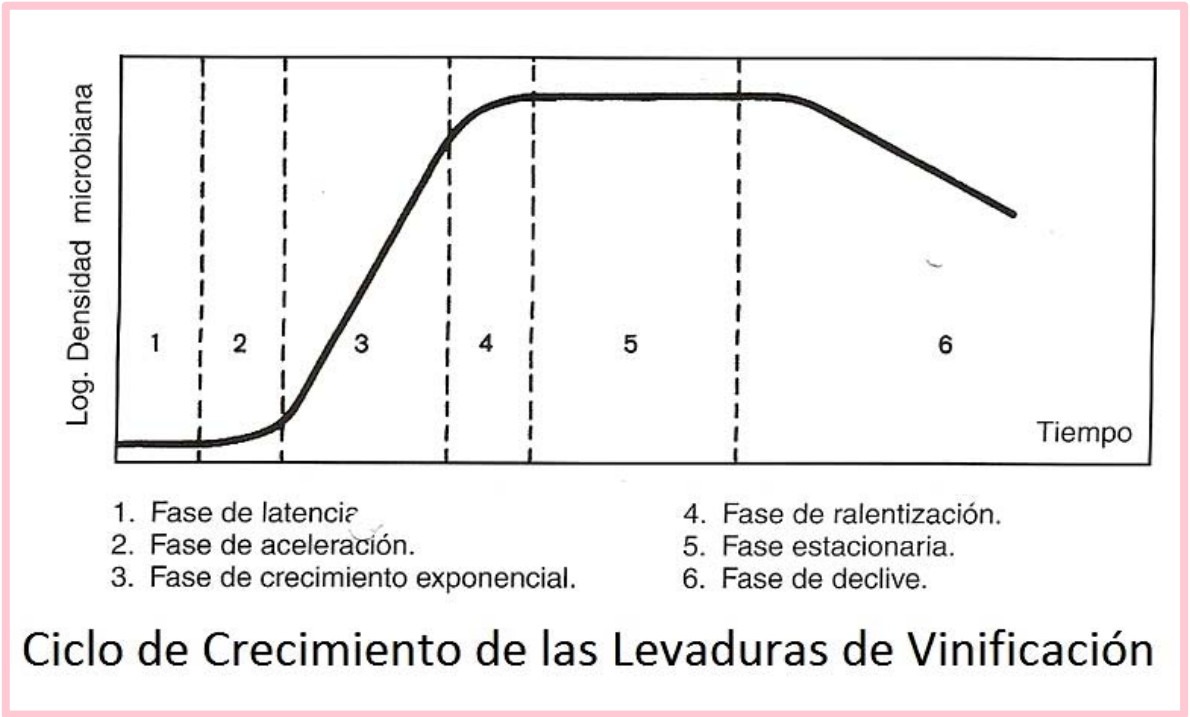


Ilustración 4 - Ciclo de Crecimiento de las Levaduras de Vinificación

Acidez (pH)

La fermentación alcohólica se produce por medio de las levaduras cuando se encuentran en entornos sin aire, descomponiendo el azúcar en alcohol y gas carbónico. El pH es un factor

limitante durante este proceso ya que las levaduras se ven afectadas por el ambiente en el que se desarrollan.

Para que las levaduras puedan desarrollarse, la acidez optima del medio debe tener un pH entre 3'5 y 5'5. No se pueden desarrollar en ambientes con valores por debajo de 2'8 por lo que no se podrá llevar a cabo la fermentación.

[Ph]	12 horas	24 horas	48 horas	4 días	8 días
Sombrero	3'6	3'7	4'0	4'1	4'1
Mosto	3'5	3'6	3'6	3'6	3'5
Fondo	3'3	3'3	3'3	3'4	3'6

Tabla 1- Ejemplo de medidas de pH en los distintos estratos de un depósito

En cuanto a la calidad del vino, el pH influye en la evolución del color del vino con el paso del tiempo. Además, define la velocidad de oxidación y deterioro de la calidad desde un punto de vista sensorial: a mayor pH, mayor riesgo de oxidación del vino durante su conservación. Para vinos con mucha acidez (< 3'5) se hace imprescindible la fermentación maloláctica.

Temperatura

Para un correcto desarrollo de las levaduras la temperatura debe establecerse entre los 20-30ºC. Lo más importante es que la temperatura no descienda de los 15ºC ya que a temperaturas más bajas se detiene la fermentación. Además, tampoco es conveniente sobrepasar los 30ºC ya que morirían las levaduras y por consiguiente también detendría la fermentación.

[ºC]	12 horas	24 horas	48 horas	4 días	8 días
Sombrero	18	30	34	34	30
Mosto	15	32	36	36	32
Fondo	15	17	20	22	26

Tabla 2 - Ejemplo de medidas de temperatura en los distintos estratos de un depósito

Por otro lado, la temperatura es un factor clave en la fermentación en cuanto a la duración del proceso. Existe una relación directa entre la cantidad de azúcar que pueden transformar las levaduras (y como consecuencia el grado que se puede alcanzar) con la temperatura del medio. Con una temperatura mayor la fermentación se lleva a cabo más rápido, pero eso conlleva que se alcance un grado de alcohol menor.

La siguiente tabla muestra los datos obtenidos de una bodega a modo de ejemplo en los que se puede observar la duración de la fermentación de un mismo proceso a tres temperaturas diferentes (se considera la fermentación finalizada cuando la densidad alcanza los 992-994 g/L).

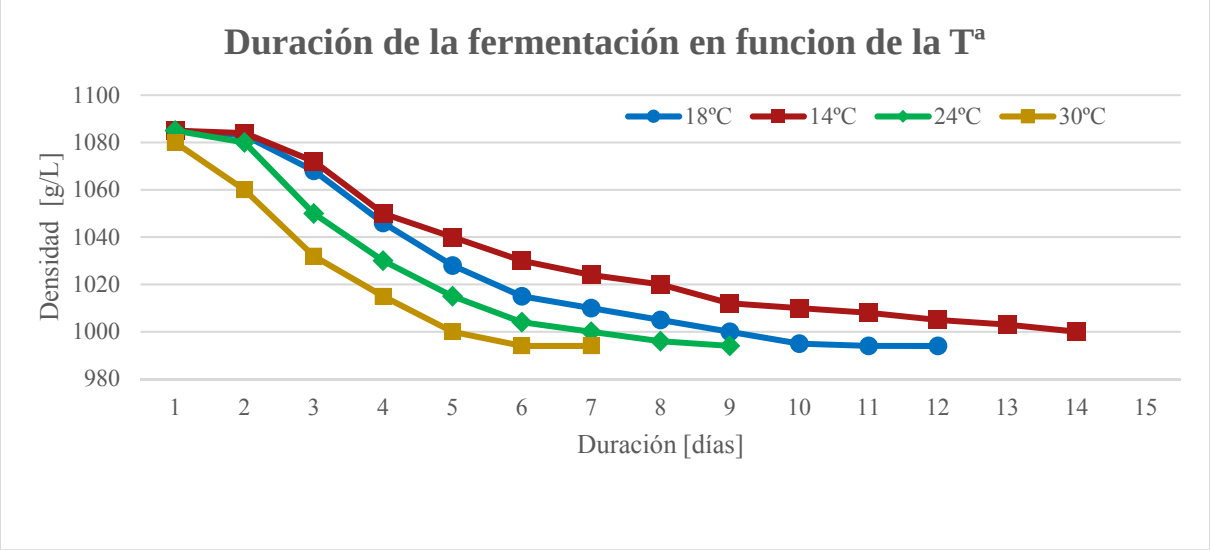


Ilustración 5 - Ejemplo de la evolución de la densidad en función de la temperatura durante la fermentación

Además de en el grado de alcohol, la temperatura influye en el sabor y color final del vino. Para obtener vinos ligeros y afrutados hay que considerar temperaturas más bajas. Por otro lado, si se desea un color más intenso y un mayor cuerpo, la temperatura deberá estar ligeramente por debajo de los 30ºC.

Densidad

El fin de la fermentación alcohólica se controla a través de la densidad. Midiendo sus valores podemos obtener de una manera bastante exacta la cantidad de azúcar que contiene el mosto, y con ello predecir el grado que tendrá el vino. Durante este periodo se mide 2 o 3 veces al día. Analizando sus valores se suelen programar los remontados. De esta forma, si se desea disminuir el valor de la densidad se programarán más remontados, ya que mediante la aireación las levaduras obtienen el oxígeno necesario para multiplicarse.

[g/L]	12 horas	24 horas	48 horas	4 días	8 días
Sombrero	1080	1062	1040	995	995
Mosto	1080	1078	1062	1022	1005
Fondo	1080	1080	1080	1025	1005

Tabla 3 - Ejemplo de medidas de densidad en los distintos estratos de un depósito

Se puede considerar que la fermentación alcohólica ha llegad a su fin cuando se obtienen valores de densidad sobre 992-994 g/L (algo menos si se trata de vino blanco). Cuanto más baja sea la densidad, mayor alcohol tendrá el vino.

Otros parámetros

En el caso de la fermentación maloláctica, la cual se lleva a cabo después de la alcohólica, también se lleva a cabo el control de la temperatura. Sin embargo, en este proceso

son las bacterias presentes de manera natural en la propia uva las encargadas de realizar la transformación del ácido málico en ácido láctico.

A diferencia de la fermentación alcohólica, en la maloláctica la temperatura óptima para el desarrollo de las bacterias es de 20ºC, pudiéndose llevar a cabo en un rango de 19-23ºC.

El principal objetivo de la fermentación maloláctica es el reducir la acidez, lo que puede durar meses, pero se considera finalizada cuando el ácido málico desciende hasta valores de 0'20-0'30.

2.4.3 CAN bus

El bus CAN (Controlled Area Network) es un protocolo de comunicaciones basado en una topología bus, es decir, los nodos se interconectan entre sí a través de un único bus. Fue presentado por primera vez en 1986 por la empresa Bosch en la Sociedad de Ingenieros de Automoción (SAE) para la transmisión de mensajes entre unidades de control electrónicas dentro del automóvil. Esto supuso un enorme impacto en la industria del automóvil ya que los coches cada vez se volvían más complejos con un mayor número de dispositivos a controlar. De esta manera se consiguió una importante reducción en peso y coste, tanto de cableado como de sensores utilizados.

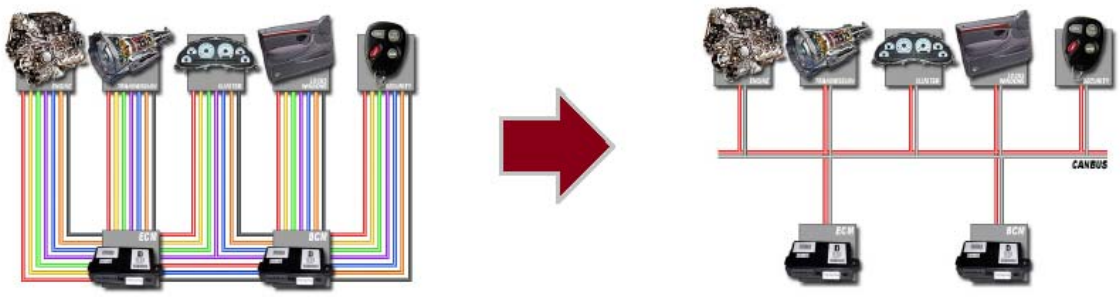


Ilustración 6 - Conexión de los componentes de un automóvil antes y después del sistema CAN

La especificación del protocolo can está publicada en el estándar ISO 11898. En esta norma se define la capa de enlace de datos y parte de la capa física: la norma ISO 11898-1 define la capa de enlace de datos mientras que en la capa física depende de la velocidad, siendo ISO 11898-2 para alta velocidad e ISO 11898-3 para baja.

La característica principal de este sistema es que es un protocolo orientado a mensajes, es decir, el mensaje no va dirigido a ningún nodo o unidad de mando en particular. Cada mensaje tiene un identificador el cual permite reconocer a cada unidad si ese mensaje le interesa o no, y en consecuencia de ello aceptarlo.

La transmisión de información se realiza de modo diferencial entre los dos cables, cuyas señales se denominan CAN-H y CAN-L. Esta información se transmite a través de tramas de longitud limitada y una estructura definida.

Toda unidad de mando puede tanto emisora como receptora, pudiendo transmitir en el bus siempre y cuando esté libre. Cuando más de una unidad intenta transmitir al mismo tiempo se resuelve el conflicto a través de la prioridad, la cual va definida en el identificador del mensaje.

Los mensajes fluyen de manera bidireccional por el bus por dos cables trenzados, lo que permite una alta inmunidad a las interferencias electromagnéticas. La impedancia característica del bus es de 120 Ω y presenta una longitud máxima de 1000 metros para velocidades de hasta 40 Kb/s. La velocidad de transmisión máxima es de 1 Mb/s, permitiendo una longitud del bus de hasta 40 metros.

Se trata de un sistema robusto en cuanto a la consistencia en la transmisión de datos, ya que presenta una serie de mecanismos que permiten una detección y señalización de errores muy fiable. Si se produce un error, se anula el mensaje y se transmite de nuevo. Del mismo modo, si el sistema detecta un fallo en alguna unidad de mando que pudiera comprometer la integridad de la red deja fuera a dicha unidad, pero la red puede seguir operativa.

2.4.4 Placa ChipKit MAX32

En los últimos años, Arduino ha supuesto una gran revolución en el mundo de la electrónica gracias a su plataforma de hardware de código abierto y sus placas de bajo coste en un entorno de desarrollo basado en el lenguaje de programación Processing.

Arduino nació de la necesidad de un profesor, de un dispositivo de bajo coste que funcionase en cualquier sistema operativo. A diferencia de otros microcontroladores, Arduino está sustentado por la filosofía ‘open source’, lo que permite que tanto el hardware como el software sean de acceso público. Esto permite que tanto su diseño como distribución sea libre y pueda emplearse en cualquier proyecto sin necesidad de adquirir ninguna licencia.

Este éxito ha provocado que en el mercado del open hardware los fabricantes estén trabajando por entrar en el negocio que inició Arduino con Atmel. Otros fabricantes de microcontroladores como Intel o Microchip han sacado sus propias placas con el mismo concepto que Arduino, como es el caso de Chipkit de Microchip y en concreto la Max32: clon de la placa Arduino Mega.

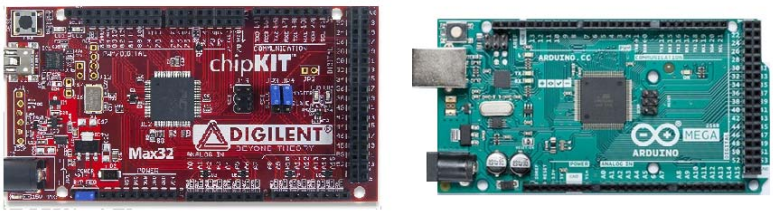


Ilustración 7 - Placas de desarrollo chipKit Max32 y Arduino Mega

Microchip ha decidido avanzar sobre este mercado con una plataforma de código abierto que es compatible con el hardware y el software de Arduino, pero con ciertas ventajas. Presenta un controlador Microchip PIC32, un mayor número de entradas/salidas, y un mayor número de puertos de comunicaciones entre otras cosas.

Para una total compatibilidad, ChipKit ha creado una versión modificada del IDE de Arduino llamada MPIDE, compatible tanto para las placas de Arduino como las de Chipkit.

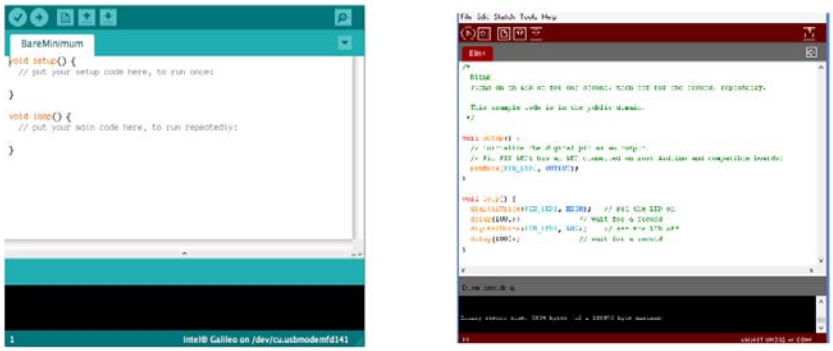


Ilustración 8 - Entornos de desarrollo IDE de Arduino y MPIDE de ChipKit

Las últimas actualizaciones del entorno de desarrollo IDE de Arduino, incorporan ya la compatibilidad con ChipKit y algunas de sus librerías de uso más frecuente.

2.5 NORMAS Y REFERENCIAS

2.5.1 Disposiciones legales y normativa aplicada

- UNE 157001:2014 "Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico".
- UNE 21302 "Vocabulario Electrotécnico".
- UNE 20-050-93 "Códigos para el marcado de resistencias y condensadores".
- Reglamento Electrotécnico de Baja Tensión.
- Estándar CANbus ISO 11898.

2.5.2 Programas de cálculo

- MySQL Workbench 8.0: es una herramienta visual para la creación y gestión de bases de datos. Proporciona múltiples herramientas para la configuración de los servidores, administración de usuarios, modelado de datos, etc, todo bajo el lenguaje de programación SQL. Está disponible para Windows, Linux y Mac OS.
- Microsoft Visual Studio 2010: es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows. Soporta múltiples lenguajes de programación, como VB.NET con el cual se ha desarrollado la aplicación de este proyecto.
- ChipKIT MPIDE: (Multi-Platform Integrated Development Environment) es un software gratuito dedicado a programar tarjetas compatibles con ChipKIT. Está basado en el IDE de Arduino.
- AutoCAD: es un software de diseño asistido por ordenador utilizado para dibujo 2D y modelado 3D.
- Microsoft Excel: es una aplicación de hojas de cálculo que forma parte de la suite de oficina Microsoft Office. Permite de forma eficaz manipular, analizar y representar datos. Soporta el lenguaje de programación VBA (Visual Basic para Aplicaciones), permitiendo realizar rutinas y macros a través de la propia aplicación o a través de otras, como puede ser Visual Studio.

2.5.3 Bibliografía

- FLORES MARTÍ, LUIS. Defectos organolépticos del vino. ¿Cuáles y por qué?, 2018.
- HIGALGO, JOSE. Tratado de Enología, 2011.

- FUIDIO BLASCO, FRANCISCO. Automatización del control de fermentación en bodega, 2013.
- PETROUTSOS, EVANGELOS. La Biblia de Visual Basic .NET, 2002.
- CLARK, DAN. Introducción a la programación orientada a objetos con Visual Basic .NET, 2003.
- GILFILLAN, IAN. La Biblia de MySQL, 2003.
- JOYANES, LUIS. ZAHONERO MARÍNEZ, IGNACIO. Programación en C,2005.
- EQUISOAIN LOZANO, DANIEL. Arduino Práctico, 2016.
- SAEZ LOPEZ-DAVALILLO, TERESA. Sistema distribuido de trazabilidad mediante comunicación Canbus, 2014.
- VICENTE GUERRERO, LUIS MARTÍNEZ. Comunicaciones Industriales, 2010.

2.5.4 Otras referencias

Los siguientes enlaces web permanecen activos a septiembre de 2018. Es posible que para consultas posteriores a esta fecha algunos enlaces puedan no estar disponibles.

Proceso de vinificación

- http://www.catadelvino.com
- https://www.devinosyvides.com.ar
- urbinavinos.blogspot.com
- www.aprenderdevino.es
- http://www.diccionariodelvino.com/

Can BUS

- https://petrolheadgarage.com/Posts/caracteristicas-de-un-sistema-can-bus/
- https://www.can-cia.org
- https://www.kvaser.com/

- http://www.motorpasionfuturo.comcoches-hibridos-alternativos/can-bus-como-gestionar-toda-la-electronica-del-automovil
- http://www.instrumentacionycontrol.net

Visual Basic.NET

- docs.microsoft.com
- www.forosdelweb.com
- www.lawebdelprogramador.com
- bytes.com

ChipKIT y programación

- forum.arduino.cc
- aprendiendoarduino.wordpress.com
- www.luisllamas.es
- saber.patagoniatec.com
- https://reference.digilentinc.com/chipkit_max32/refmanual

Transceptor CAN

- http://ww1.microchip.com/downloads/en/DeviceDoc/21667E.pdf

Descargas

- https://www.mysql.com/products/workbench/
- https://store.digilentinc.com/mpide-multi-platform-integrated-development-environment-download-only/
- https://visualstudio.microsoft.com/es/vs/older-downloads/

2.6 DEFINICIONES Y ABREVIATURAS

- Grado Beamé: unidad que se utiliza en la bodega para medir la cantidad de azúcar del mosto o vino en función de su densidad. También se denomina grado dulce

- Grado Brix: mide la madurez de las uvas. Es la unidad de la escala que mide el azúcar en el mosto o en el vino. 1'8º Brix corresponden a 1º Baumé.

- Raspón: es la estructura vegetal del racimo de uva. También se denomina raspa o escobajo.

- Hollejo: piel que envuelve la pulpa o parte carnosa de la uva. A los hollejos también se les llama orujos.

- Gas carbónico: gas generado en grandes cantidades durante la fermentación alcohólica. Aporta frescor y produce un leve cosquilleo en la lengua.

- Sangrado: procedimiento por el cual se separa el hollejo del mosto, escurriéndolo, de modo que el líquido cae y los residuos sólidos se retienen.

- Filosofía "open source": se refiere a la filosofía de código abierto. Se trata de un modelo de desarrollo software basado en una colaboración abierta de los usuarios. El término "libre", además de referirse al poder adquirir el software de manera gratuita, hace referencia a la libertad de poder modificar la fuente del programa sin restricciones de licencia.

- CAN: acrónimo de Controller Area Network.

- IDE: se refiere al entorno de desarrollo integrado. Procede del inglés "Integrated Developed Environment". Es la aplicación informática que proporciona al usuario programador servicios integrales para facilitarle el desarrollo de software.

- LocalHost: es el nombre que se usa para designar el ordenador o dispositivo que se está utilizando en un momento determinado. Se define como dispositivo local o servidor local. Tiene asignada la dirección IP 127.0.1. Actúa como una dirección estándar, pero su acceso solo está disponible en el ordenador local y no requiere de conexión a internet.

- CS: es el acrónimo de checksum. Dentro de una trama, se trata de una suma de verificación cuyos operandos son los demás elementos de la trama. Se emplea para detectar cambios durante la transmisión de mensajes para así proteger la integridad de los datos del mensaje. Se verifica que no haya discrepancias entre los valores obtenidos al hacer una comprobación inicial y otra final tras la transmisión. En esta aplicación concreta, es el resultado de realizar la XOR lógica entre todos los elementos de la trama.

Se comprueba que los datos están íntegros si la operación XOR de todos los elementos incluidos el CS da como resultado cero.

- Firmware: es un programa informático que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo.

- Licencias Creative Commons: son una herramienta legal de carácter gratuito que permite a los usuarios usar obras protegidas por derecho de autor sin solicitar el permiso del autor de la obra.

- Transceiver: o transceptor, es un dispositivo que cuenta con un transmisor y un receptor que comparten el mismo circuito o se encuentran integrados dentro de la misma caja.

- Shield: las shields son placas de circuitos modulares que se montan unas encima de otras para aportar funcionalidades extra a otras placas.

- Bit stuffing: se trata de un mecanismo para evitar la desincronziación entre unidades de mando en la transmisión de tramas. Cuando se suceden una serie de *n* bits del mismo valor, se introduce un bit extra de valor contrario. Como consecuencia, las unidades de mando que reciban el mensaje descartarán un bit posterior a *n* bits iguales.

- AENOR: Asociación Española de Normalización y Certificación.

- Data Logger: También denominado registrador de datos, es un dispositivo electrónico que registra datos en el tiempo o en relación a la ubicación por medio de instrumentos y sensores propios o conectados externamente. Generalmente están basados en microcontroladores.

2.7 REQUISITOS DE DISEÑO

2.7.1 Red de nodos

- ✓ Para la programación de las placas se ha previsto el desarrollo un único firmware, con el cual cada placa podrá trabajar con funciones de nodo maestro o de nodo esclavo. El programa ejecutado por todas las placas será el mismo.

- ✓ La identificación de cada nodo, será llevada a cabo por el usuario. Esto se realizará asignando una dirección a cada nodo vía hardware. Esta dirección será la que definirá que nodo es el maestro y cual los esclavos.

- ✓ Únicamente podrá haber un nodo maestro, que servirá de unión entre la red de nodos y la aplicación en el PC. Este nodo será el encargado de recoger las ordenes de la aplicación y transmitirlas por la red de nodos. A su vez, recogerá la información transmitida por la red y la enviará a la aplicación. Si este nodo fallase se podrá sustituir por un nodo cualquiera del sistema, asignando a este nodo la dirección del nodo maestro.

- ✓ Los pines habilitados como canales de entrada estarán descritos en la aplicación, de tal forma que no todos los pines de la placa estarán disponibles y cada uno tendrá definido su cometido (entrada analógica, digital, serie, etc).

- ✓ El código del firmware estará cerrado para el usuario, ya que no es necesaria ninguna modificación por su parte.

2.7.2 Aplicación

- ✓ Para el acceso a la aplicación será necesario un usuario y una contraseña, los cuales se podrán gestionar previamente desde el software propio de la base de datos (MySQL Workbench). Además, se indicará la ubicación de la base de datos: si es un servidor local o un servidor externo.

- ✓ El entorno de la aplicación tendrá un diseño sencillo e intuitivo. De esta manera el usuario podrá familiarizarse con la aplicación rápidamente.

- ✓ Al tratarse de un prototipo, se permitirá la configuración de hasta 10 nodos, sin contar el nodo maestro. Para cada nodo se podrán configurar hasta 8 entradas analógicas, 8 entradas digitales, 2 bloques de entradas de 8 bits y 2 puertos de comunicación vía serie. Además, para cada entrada se podrán programar las horas a las que se desea que tome una muestra, con un máximo de 10 horas.

- ✓ A cada entrada de cada nodo, le corresponderá una tabla dentro de la base de datos. Dentro de esta tabla se almacenará el valor y la hora de cada muestra programada. Cada nodo tendrá su propia base, y cada entrada tendrá su tabla dentro de esta base. Tanto el nombre de las bases como de las tablas se podría configurar desde la aplicación.

- ✓ Los valores de las muestras tomadas no se introducirán directamente en la base de datos, sino que se almacenarán en un buffer ubicado en el propio PC. En este buffer se guardará para cada muestra el nodo y entrada de la que procede, su valor y la fecha y hora de la muestra. Mientras exista conexión con la base de datos, los registros del buffer se irán volcando automáticamente dentro de ella. Esto se realizará para que ante un fallo de la conexión no se pierdan los datos. Si esto ocurriese se guardarían los datos en el buffer y en el momento en el que la conexión volviese, se volcarían a la base de datos.

- ✓ Desde la propia aplicación se podrá acceder a la visualización de las bases de datos y las tablas. Además, se podrán representar los datos en gráficas para su mejor interpretación.

- ✓ Como la lectura de las entradas serán valores comprendidos entre 0 y 255 (0-3.3V), se podrán configurar los valores de la recta de conversión característica de cada sensor conectado. Esta conversión se realizará antes de que se introduzca el valor en la base de datos. De esta manera el usuario dispondrá de los valores ya adaptados para su visualización.

2.7.3 Base de datos

- ✓ Los datos recogidos se almacenarán en una base de datos MySQL. Esta base podrá estar localizada en el propio equipo o en un servidor externo. En la aplicación se introducirá la IP donde está ubicada la base de datos.

- ✓ Mediante el software propio de la base de datos se podrá gestionar los usuarios, creando nuevos usuarios o modificando los permisos de los ya existentes.

- ✓ Dentro de la base de datos se guardará cada nodo como una nueva base; y dentro de ésta, una tabla para cada entrada. Los campos de cada tabla serán el valor de la muestra ya convertido a unidades del parámetro que se está midiendo, y la fecha y hora a la que se ha tomado la muestra.

2.7.4 Costes del proyecto

- ✓ El software utilizado será de libre uso y el hardware de bajo coste.

- ✓ El proyecto estará bajo licencia Creative Commons.

2.8 ANÁLISIS DE SOLUCIONES

2.8.1 Placa base

La misión de los nodos en el sistema es la de medir los valores de los distintos parámetros del proceso de fermentación (densidad, temperatura, pH, etc) y transmitirlos a través de la red al nodo maestro, que es el encargado de enviar esa información a la aplicación para su registro en el buffer y su almacenamiento posterior en una base de datos.

Este prototipo de sistema no está ligado específicamente al ámbito de un proceso concreto de vinificación, en una bodega concreta (el prototipo de sistema podría utilizarse incluso en cualquier proceso industrial que necesite una monitorización y registrado de las condiciones de proceso, por ejemplo, con propósitos de trazabilidad). Puesto que en cada bodega concreta dispondrán de diferente equipamiento, se necesitará una placa capaz de interactuar con una gran variedad de sensores que nos aporten salidas digitales, analógicas, vía serie, etc.

Hoy en día en el mercado existen multitud de controladores y placas dedicadas a sistemas embebidos, pero de todas ellas las más populares son las desarrolladas por Arduino junto a la empresa de microcontroladores Atmel.

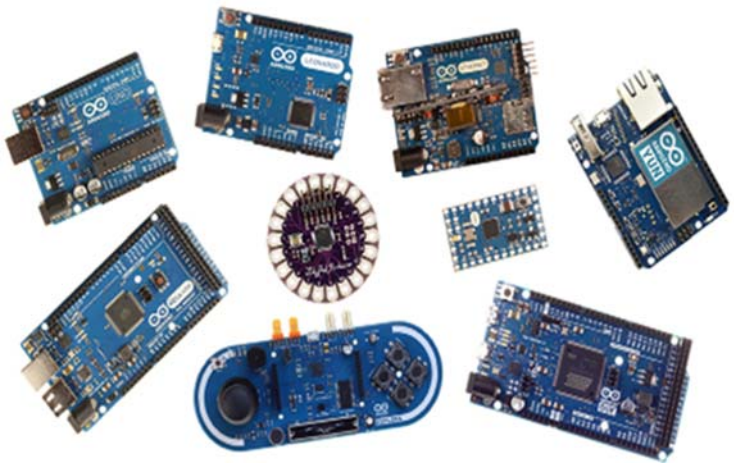


Ilustración 9- Diferentes tipos de placas Arduino

Las ventajas de estas placas frente a otros sistemas son básicamente su bajo coste y su software de código abierto. Además, funcionan en todos los sistemas operativos y la distribución de su licencia es libre.

Sin embargo, muchas empresas han querido establecer su sitio en este mercado, como son los fabricantes de microcontroladores Intel y Microchip. Éste segundo junto a Digilent han desarrollado una línea de placas ChipKIT montando en ellas microcontroladores de 32 bits, frente a los de 8 bits de Atmel que monta Arduino en sus placas. Gracias a esta considerable potencia, las placas ChipKIT se han posicionado en el mercado debido también a la total

compatibilidad que presentan con el hardware y el IDE de Arduino, desde diferentes shield hasta códigos y librerías.

A continuación, se va a realizar una tabla comparativa de la placa Arduino Mega, y su equivalente ChipKIT Max32.

	Arduino Mega 2560	**ChipKIT Max 32**
Microcontrolador	ATMega2560 - 8bit	Microchip PIC32 - 32bit
Velocidad del reloj	16 MHz	80 MHz
Tensión de alimentación	7-12 V	7-12 V
E/S Digitales	54 (15 PWM)	58 (5 PWM)
Entradas analógicas	16	16
Corriente de E/S	20 mA	21 mA
Memoria Flash	256 K	512 K
SRAM	8 K	128K
EEPROM	4K	4K
RTTC (Real time clock)	-	Sí
Ethernet	-	Sí
Controlador CAN	-	x2
Comparadores	x1	x2
Temporizadores	8/16 Bit	16/32 Bit
I2C	x1	x5
UART	x4	x4
SPI	x1	x2

Tabla 4 - Comparativa de características entre Arduino Mega y ChipKIT Max32

Como se puede observar, la Max32 reúne todas las ventajas que presenta el sistema Arduino con unas notables mejoras en cuanto a potencia y capacidad de procesamiento. Además, aprovechando los dos controladores CAN que lleva incorporado nos van a permitir desarrollar un sistema robusto y fiable. Es por ello que **se ha elegido la ChipKIT Max32 como placa base para gobernar los nodos de nuestro sistema.**

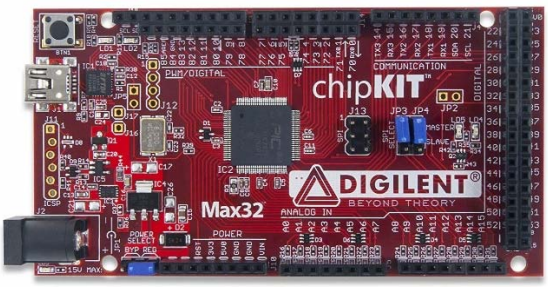


Ilustración 10 - Placa chipKIT Max32

2.8.2 Bus de comunicaciones para la red de nodos

Una vez definidas las placas que van a formar los nodos del sistema, el siguiente paso es establecer el bus por el cual se va a realizar la transmisión de información entre nodos. Hoy en día existen numerosos buses industriales, todos ellos destinados a sustituir la conexión punto a punto entre los nodos de un proceso industrial.

Para la elección del bus de campo para este proyecto, será necesario estudiar los distintos aspectos que presenta cada tipo de bus como son el autodiagnóstico, la seguridad o la robustez. A continuación, se describirán varios buses de campo para analizar cuál es el más conveniente para nuestro proyecto.

AS-I

AS-Interface es un sistema de bus de campo estandarizado a nivel mundial (EN 50295, IEC62026-2) que conecta sensores y actuadores mediante un único cable bifilar característico de este sistema, a través del cual se transmiten tanto datos como potencia. Los elementos de este sistema son un maestro, esclavos, el cable Así y una fuente de alimentación.

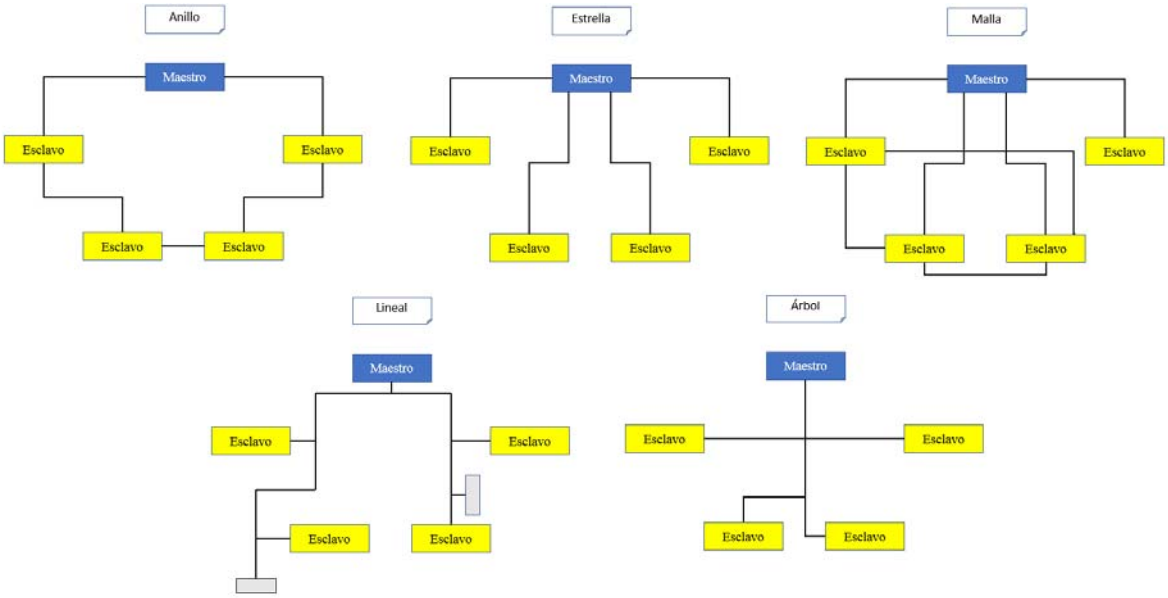


Ilustración 11 - Topologia de las redes ASi

Dentro de la red cada esclavo tiene asignada una dirección única a través de la cual el maestro gestiona la comunicación con los esclavos. A diferencia de otros sistemas de bus, la red AS-i se configura de forma automática ya que el usuario no necesita realizar ningún ajuste como derecho de acceso, velocidad de red, tipo de trama, etc.

Ilustración 12 - Cable AS-i

Permite hasta 62 esclavos por red con una distancia máxima de 100m, o 300m si se dispone de repetidores.

Profibus

Es un estándar de comunicaciones desarrollado a finales de los años 80 por Bosch, Klöckner Möller y Siemens entre otras. Fue confirmada como norma europea en 1996 como EN50170. Funciona bajo la metodología maestro- esclavo para niveles desde sensor-actuador procesos mucho más complejos.

Profibus tiene varias tecnologías de transmisión, como la RS-485 con un par trenzado y apantallado, o incluso fibra óptica. Emplea además un terminador de bus para absorber las señales de la línea evitando que reboten y vuelvan a ser recibidas por los nodos de la red.

Ilustración 13 - Cable y conector Profibus

Este protocolo permite un máximo de 127 nodos, con una distancia de hasta 24 Km si la transmisión se realiza a través de fibra óptica.

Modbus

Es un protocolo de comunicaciones basado también en la arquitectura maestro/esclavo. Fue diseñado en 1979 por Modicon. Hoy en día es el protocolo que goza de mayor disponibilidad para la conexión de dispositivos electrónicos industriales.

Es fácil de implementar en todo tipo de topologías de red ya que requiere poco desarrollo, además de público y gratuito. El cable presenta un par aislado, trenzado y apantallado.

Ilustración 14 - Cable Modbus

Cada segmento del sistema permite hasta 32 nodos con una distancia máxima de 500 m por segmento.

Ethernet/IP

Es un protocolo de red a nivel de aplicaciones de automatización industrial. Está basado en los protocolos estándar TCP/IP por lo que no tiene ni límite de nodos ni límite de distancia. Utiliza los ya conocidos hardware y software Ethernet para configurar, acceder y controlar los distintos dispositivos dentro de una red. Está definido en la norma IEEE 802.3.

En este bus de campo se emplea la tipología estrella, lo que facilita la detección de errores en el cableado. Emplea un bus 10/100 base T formado por el conector RJ45 y el cable Cat 5E.

Ilustración 15 - Cable Ethernet

CAN bus

El bus CAN (Controlled Area Network) es un protocolo de comunicaciones basado en una topología bus. Fue presentado por primera vez en 1986 por la empresa Bosch en la Sociedad de Ingenieros de Automoción (SAE) para la transmisión de mensajes entre unidades de control electrónicas dentro del automóvil. Esto supuso un enorme impacto en la industria del automóvil ya que los coches cada vez se volvían más complejos con un mayor número de dispositivos a controlar. De esta manera se consiguió una importante reducción en peso y coste, tanto de cableado como de sensores utilizados. La especificación del protocolo can está publicada en el estándar ISO 11898.

Es un protocolo orientado a mensajes, es decir, el mensaje no va dirigido a ningún nodo o unidad de mando en particular. Cada mensaje tiene un identificador el cual permite reconocer a cada unidad si ese mensaje le interesa o no, y en consecuencia de ello aceptarlo.

La transmisión de información se realiza de modo diferencial entre dos cables trenzados, lo que permite una alta inmunidad a las interferencias electromagnéticas. La impedancia característica del bus es de 120 Ω y presenta una longitud máxima de 1000 metros para velocidades de hasta 40 Kb/s. La velocidad de transmisión máxima es de 1 Mb/s, permitiendo una longitud del bus de hasta 40 metros.

Ilustración 16 - Cable CANbus

Se trata de un sistema robusto en cuanto a la consistencia en la transmisión de datos, ya que presenta una serie de mecanismos que permiten una detección y señalización de errores muy fiable. Si se produce un error, se anula el mensaje y se transmite de nuevo. Del mismo modo, si el sistema detecta un fallo en alguna unidad de mando que pudiera comprometer la integridad de la red deja fuera a dicha unidad, pero la red puede seguir operativa. Según la tipología admite hasta 64 nodos con un máximo de 127 esclavos

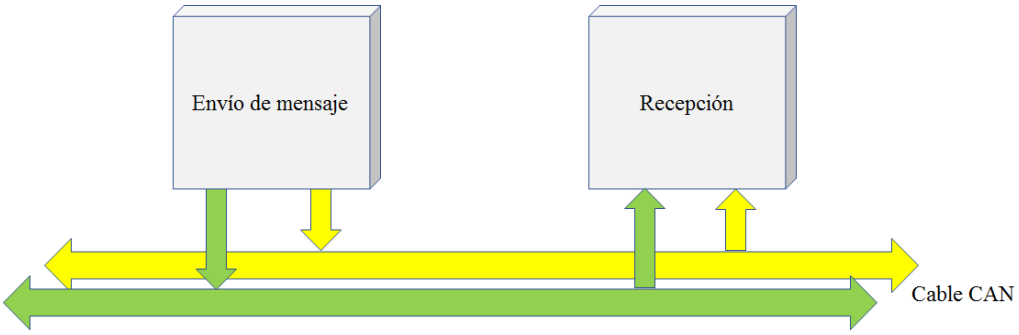


Ilustración 17 - Transmisión de mensajes en red CAN

Valorando toda la información anterior junto con los elementos del proyecto ya elegidos anteriormente**, se ha decidido desarrollar la red de nodos mediante CAN bus**. Este bus presenta la conexión más favorable para las placas ChipKIT Max 32 ya que éstas tienen integrados dos controladores CAN. Además, el hecho de que este protocolo permita un elevado número de nodos y sea robusto en cuanto a la consistencia en la transmisión de datos, hacen que idóneo para este proceso.

2.8.3 Aplicación en PC

Para la configuración del funcionamiento de la red de nodos y la base de datos, se va a desarrollar una aplicación que funcione bajo PC. Esta aplicación se va a realizar mediante programación orientada a objetos, ya que el objetivo es obtener una herramienta visual e intuitiva para el usuario.

Hoy en día existe una gran variedad de lenguajes de programación orientada a objetos, cada uno con sus ventajas e inconvenientes. Por eso, previamente es necesario definir qué herramientas son necesarias que tenga el lenguaje de programación elegido, para poder llevar a cabo las funcionalidades que se desean que tenga la aplicación:

- Lenguaje intuitivo para que la dificultado de la curva de aprendizaje no sea elevada. No tiene que ser necesario profundizar demasiado dentro del lenguaje para poder desarrollar la aplicación.

- Suficientes librerías disponibles. Muchas funcionalidades de la aplicación se van a basar en la comunicación con bases de datos, comunicación serie con el nodo maestro, o interactuar con archivos Excel. Por ello, interesa que se disponga de librerías y herramientas ya creadas para facilitar el desarrollo de la aplicación.

- Comunidad de usuarios activa. Si se trata de un lenguaje que tiene una gran comunidad de usuarios es posible que muchas funcionalidades o problemas que pueda presentar la aplicación ya estén resueltos, evitando atascos durante el desarrollo de la aplicación.

Herramienta	C Builder	VB.NET	Visual Basic 6	Borland Jbuilder	Nerbeans
Lenguaje	C++	Visual Basic	Visual Basic	Java	Java, C++

Tabla 5 - Lenguajes de programación orientada a objeto de distintos entornos

Estudiando las distintas opciones, **se va a desarrollar la aplicación del proyecto mediante Visual Basic.NET** a través del IDE Microsoft Visual Studio 2010. Esta herramienta permite la descarga de extensiones que serán de ayuda para realizar nuestro proyecto, facilitando la conexión con bases de datos, comunicaciones serie y la manipulación de archivos Excel.

2.8.4 Base de datos

La principal consideración a la hora de elegir la base de datos sea que su uso sea libre. Existe una gran oferta de bases de datos, como Access, Oracle, MySQL,etc.

Ilustración 18 - Algunas de las principales bases de datos del mercado

El sistema Access destaca por ser gráfico. Pertenece a la suite Office de Microsoft, por lo que es exclusivo para Windows. El hecho de que sea gráfico, permite una gestión mas sencilla de los datos. Además, el usuario tiene a su disposición una gran variedad de asistentes lo que permite ahorrar mucho tiempo a la hora de crear la base de datos.

Por otro lado, otra de los sistemas mas utilizados a la hora de crear bases de datos es Oracle. Es muy utilizado en grandes empresas ya que permite gestionar bases de datos de gran

tamaño. En cuanto a su compatibilidad, es multiplataforma. Sin embargo, su principal desventaja es su elevado coste ya que no se trata de software libre. Esto hace que no sea una opción rentable si no se va a gestionar una gran cantidad de datos.

Dentro de las plataformas de uso libre, destaca MySQL. Destaca tener un uso muy sencillo e intuitivo. Además, es de código abierto y muy fácil de configurar. Una de las ventajas principales es que numerosos entornos de desarrollo tienen extensiones para trabajar de manera más sencilla con esta base de datos como es el caso de Visual Studio 10 (elegido para este proyecto). Por ello **MySQL será la base de datos elegida para nuestro proyecto.**

2.9 RESULTADOS FINALES

Ya elegidas las herramientas y elementos que van a dar forma al proyecto, en este apartado se van a desarrollar las tareas realizadas para la consecución del prototipo cumpliendo los objetivos marcados.

2.9.1 Prototipo a desarrollar: Esquema general del sistema

Para elaborar un buen vino entran en juego muchos factores. Algunos de ellos proceden de la tradición, pero otros son introducidos por las nuevas tecnologías. Incorporar nuevos avances un proceso tan tradicional como es la elaboración del vino, permite cada vez optimizar más este proceso y obtener un producto de mayor calidad.

Desafío

El desafío de este proyecto es el crear un prototipo capaz de tomar y almacenar datos sobre distintos parámetros que influyen en el proceso de fermentación del vino, dotando al proceso de cierta trazabilidad. Mediante el análisis de estos datos, el enólogo podrá ser capaz de establecer patrones y relaciones entre los datos tomados y las características del producto final. De esta manera, se podrán realizar estrategias y predicciones en futuras elaboraciones para poder obtener un vino con las características deseadas.

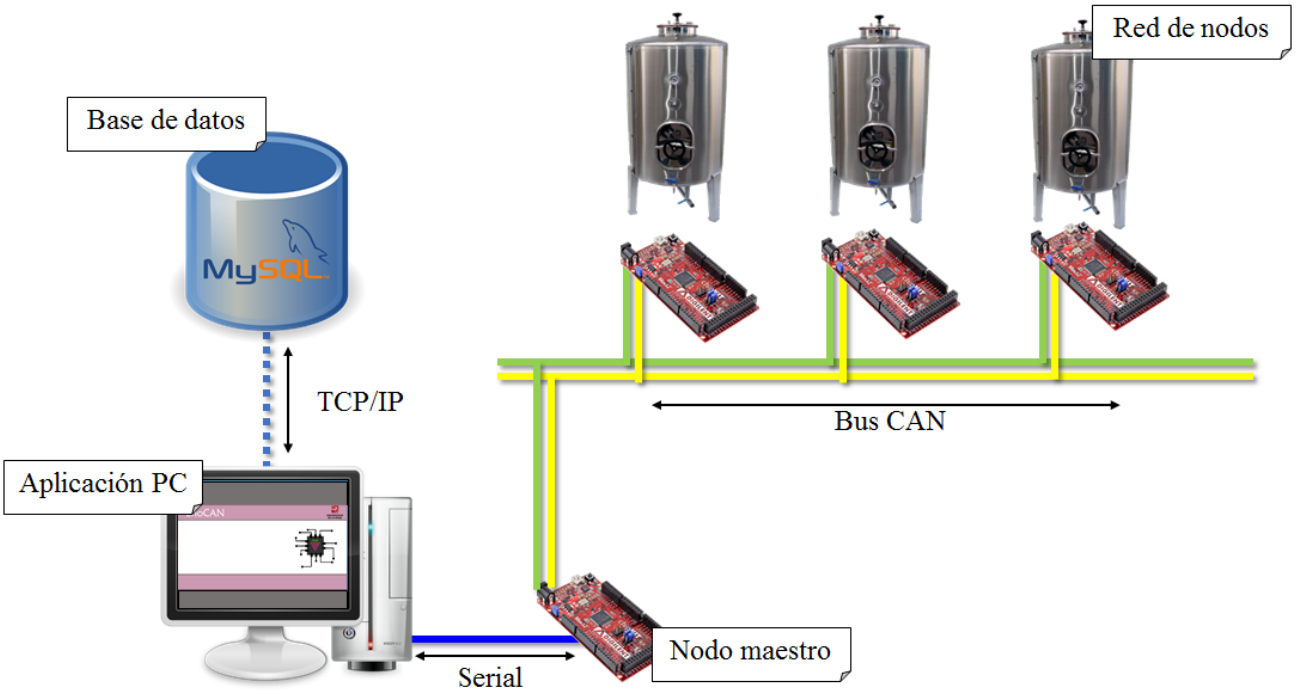


Ilustración 19 – Estructura del sistema completo

Este sistema estará formado por:

- Una base de datos en la cual se alojará la información para su disposición por el usuario

- Una aplicación en PC, mediante la cual el usuario pueda configurar la red de nodos encargada de tomar la información sobre las distintas variables del proceso. Desde esta aplicación se podrá configurar el funcionamiento de la red de nodos, y acceder directamente a la base de datos para la visualización de los datos.

- Una red de nodos CAN bus encargada de la toma de datos. Esta red estará gobernada por un nodo maestro que será el encargado de transmitir esa información a la aplicación para su almacenamiento en la base de datos.

2.9.2 Base de datos

Como ya se ha indicado, para el desarrollo de la parte de la base de datos se ha utilizado MySQL. Parte de la gestión de la base de datos como es la inserción y visualización de datos se realiza desde la propia aplicación. Sin embargo, hay otras funciones como la creación de una nueva conexión o la gestión de usuario que se han de realizar mediante MySQL Workbench.

MySQL Workbench

Se trata de una herramienta visual para la arquitectura de bases de datos. Proporciona modelado de datos, desarrollo de SQL y otras herramientas que emplearemos como la configuración del servidor y la administración de usuarios. Se trata de una herramienta de libre acceso y su descarga es gratuita desde su propia web (ver apartado "*2.4.4 Otras referencias*").

Los datos se pueden almacenar en un servidor local (LocalHost) o en un servidor remoto. Desde la aplicación se puede seleccionar la ubicación de la base de datos a través de la dirección IP, pero esta conexión debe estar previamente creada. Eso se realiza desde My SQL Workbench.

Al configurar la nueva conexión, hay que establecer que se realice a través del método TCP/IP. Para una conexión local, hay que asignar el servidor LocalHost que hace referencia al propio equipo, o la IP equivalente 127.00.1. Si se desea configurar un servidor remoto, hay que introducir la IP del servidor. Tanto en servidor local como en remoto, el puerto estándar para la conexión es el 3306. Por último, se debe asignar una contraseña al usuario 'root' del sistema, el cual tendrá todos los permisos.

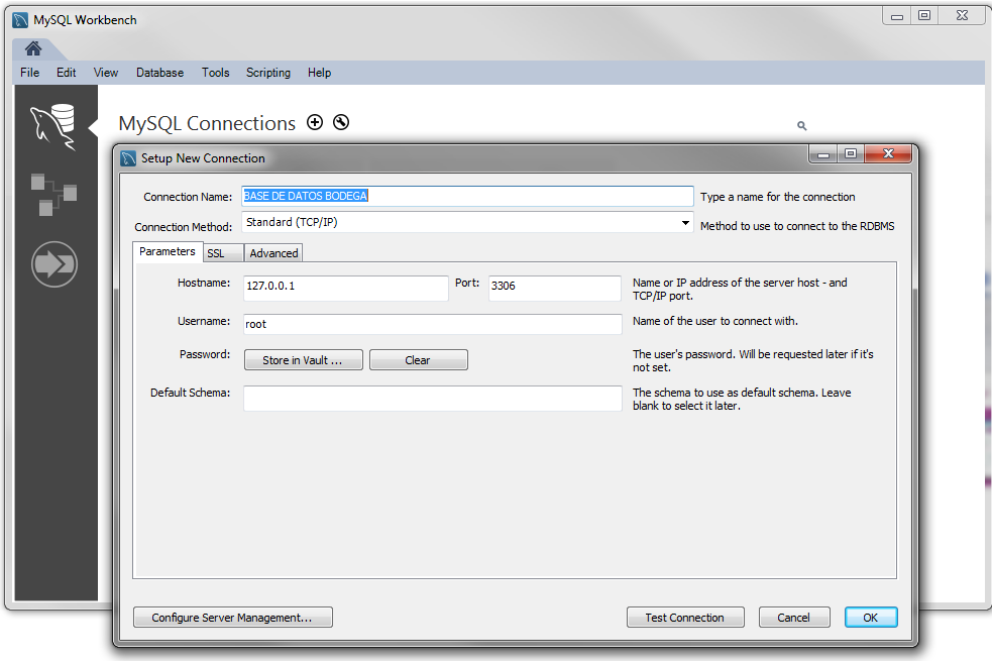


Ilustración 20 - Configuración de nueva conexión MySQL

Una vez creada la conexión, esta herramienta también permite conectarse y gestionar la base de datos. Desde aquí se crean las cuentas de usuario que permitirán el acceso a la aplicación.

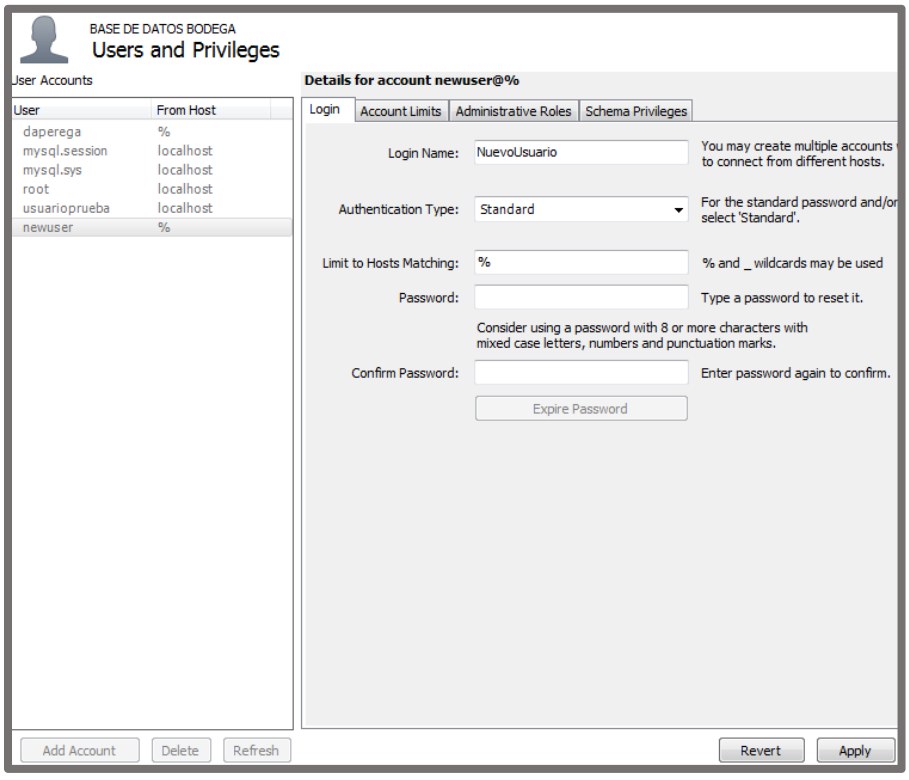


Ilustración 21 - Gestión de Usuarios

Para cada nodo se creará una base de datos, también denominada esquema. Dentro de cada esquema se guardarán las tablas asociadas a cada entrada del nodo. Los nombres de los esquemas y las tablas se establecerán desde la aplicación PC.

Cada tabla tendrá dos columnas donde se guardará el valor de la medida y la fecha en la que se tomó. Estos valores ser irán introduciendo en la base de datos a medida que el sistema los vaya capturando. En la siguiente ilustración se pueden observar las bases de datos asociadas a cada depósito, y dentro de cada una las tablas asociadas a cada parámetro.

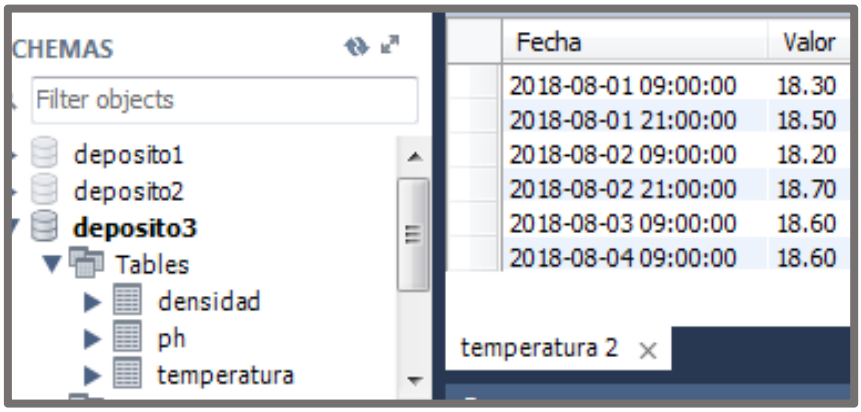


Ilustración 22 - Detalle de la representación de muestras en MySQL Workbench

Asignando a cada dato su referencia, permitirá al usuario realizar una trazabilidad exitosa del proceso. Sin embargo, el usuario no necesitará acceder a esta herramienta para la visualización de datos, ya que ese cometido se podrá realizar directamente desde la aplicación. Además de poder visualizar las tablas como se muestra en la anterior ilustración, se podrán generar gráficas para una interpretación mejor y más visual de los datos.

2.9.3 Aplicación bajo PC

Esta aplicación tal vez sea la parte más determinante del proyecto. Es el elemento de unión entre las partes del sistema ya que es la encargada de la configuración de la red de nodos para la toma de valores, y también de introducir esos valores dentro de la base de datos.

A continuación, se va a explicar el entorno de la aplicación con algunos matices sobre la programación de las funciones llevadas a cabo. El desarrollo y explicación de todas las funciones llevadas a cabo en la aplicación se encuentra en el apartado "*3. Programación de la aplicación VB.NET*". El nombre de la aplicación, EnoCAN, hace referencia a la aplicación de un sistema de comunicaciones CAN dentro de un proceso enológico.

Inicio de la aplicación

Al arrancar la aplicación aparece la siguiente ventana:

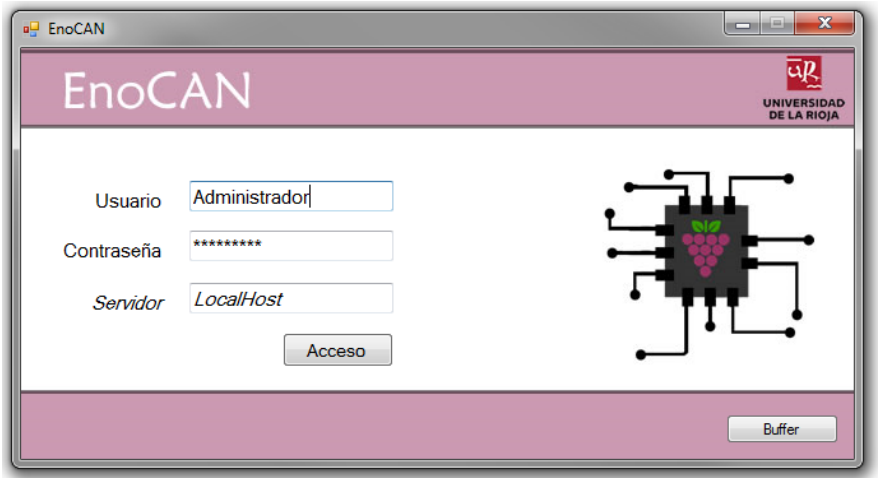


Ilustración 23 - Pantalla de inicio de EnoCAN

En ella aparecen tres campos a rellenar: un usuario, contraseña y el servidor. En ellos se debe introducir una cuenta de usuario válida registrada en la base de datos, como se ha explicado en el apartado anterior. En el campo servidor habrá que introducir LocalHost (o bien 127.0.0.1) si se trata de un servidor local, o si se trata de servidor remoto su IP correspondiente.

Modo buffer

Cuando la aplicación recibe los valores de los parámetros, éstos no se guardan directamente en la base de datos, sino que se introducen en un buffer. Se trata de un archivo Excel ubicado en el PC que la aplicación crea automáticamente. Funciona como una memoria FIFO, en la cual el primer dato que entra es el primero que sale

Cuando el sistema funciona en "modo normal", la aplicación comprueba continuamente si hay datos dentro del buffer, y si los hay, los introduce dentro de la base de datos y los elimina del buffer. A parte del valor y la fecha de la muestra, también se guarda el nodo y la entrada a la que pertenece. De esta manera la aplicación sabe en qué base de datos y tabla debe introducir la muestra.

Si mientras el sistema está funcionando se produce algún error en la conexión con la base de datos, el sistema automáticamente cambia a "modo buffer". También se puede acceder al modo buffer pulsando el botón "Buffer" de la pantalla de inicio. Esto es útil si por ejemplo no se dispone de un usuario y contraseña de acceso a la base de datos.

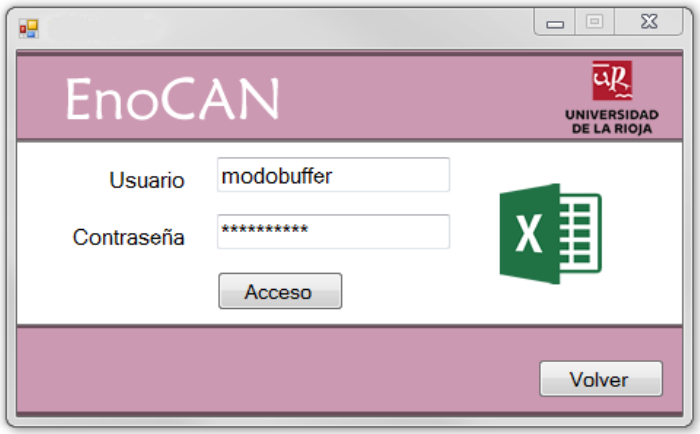


Ilustración 24 - Acceso al modo Buffer

En esta ventana habrá que introducir un usuario y contraseña el cual se le facilitará al usuario.

En el "modo buffer" el sistema no almacena los datos en la base de datos, sino que los va almacenando en el archivo Excel. La aplicación permitirá configurar los nodos y recibir muestras, pero se quedarán guardados en el buffer y no se almacenarán en la base de datos. Para volcar los datos del buffer, basta con conectarse al "modo normal" introduciendo en la ventana de inicio el usuario y contraseña válidos. De esta forma el sistema funcionará de nuevo en "modo normal".

Ventana principal

Tanto si accedemos en "modo normal" como en "modo buffer", al acceder al sistema lo primero que se tendrá que seleccionar es el número de nodos. Posteriormente en la ventana de configuración vendrá representado el modo de operación en una etiqueta. Como se ha indicado en el apartado "*2.4 Requisitos de diseño*", al tratarse de un prototipo se podrán configurar hasta 10 nodos.

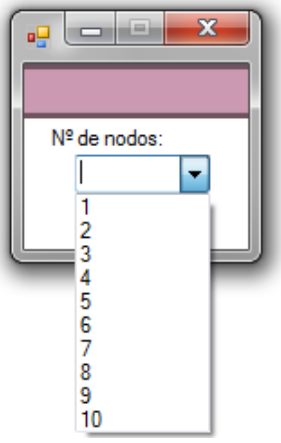


Ilustración 25 - Selección del nº de nodos

Tras seleccionar el número de nodos del sistema, aparecerá la pantalla principal para su configuración. A continuación, se van a explicar los diferentes campos que la forman.

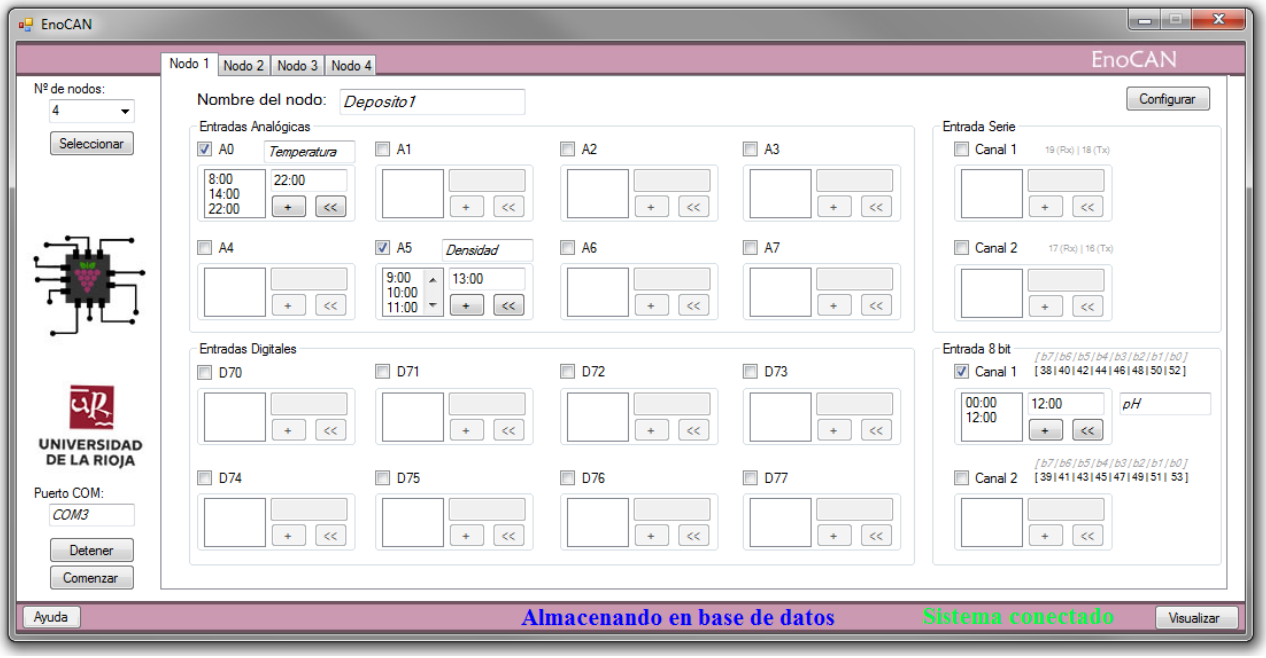


Ilustración 26 - Formulario principal de la aplicación.

- ✓ En función del nº de nodos seleccionados, aparecerá en la zona superior una lista de pestañas correspondientes a cada nodo. Dentro de cada pestaña podremos configurar el funcionamiento de cada nodo.

- ✓ En el campo "nombre del nodo" corresponde al nombre que se desea dar al esquema de ese nodo.

- ✓ Para cada nodo hay disponibles 8 entradas digitales, 8 entradas analógicas, 2 entradas de byte y 2 canales para la comunicación serie. En cada entrada viene indicado el pin al que se corresponde. Cuando se selecciona cada entrada, se desbloquea la zona donde se introducen las horas a las que se desea que se tome una medida. Además, aparece junto a la entrada un campo donde se puede introducir el nombre que se le desea dar a la tabla correspondiente a esa entrada en la base de datos.

- ✓ Para cada muestra, se recibe desde la red de nodos un valor comprendido entre 0 y 255. Pulsando el botón "Configurar", aparecerá una ventana que permitirá establecer las ecuaciones características de los sensores empleados para adecuar el valor recibido de la red de sensores a valores en unidades reales.

- ✓ Si el sistema está trabajando en "modo buffer" aparecerá un botón "Conectar" junto a la etiqueta "Almacenando en buffer". Mediante este botón se podrá acceder al

formulario de inicio, el cual permitirá conectarnos a la base de datos y trabajar en "modo normal"

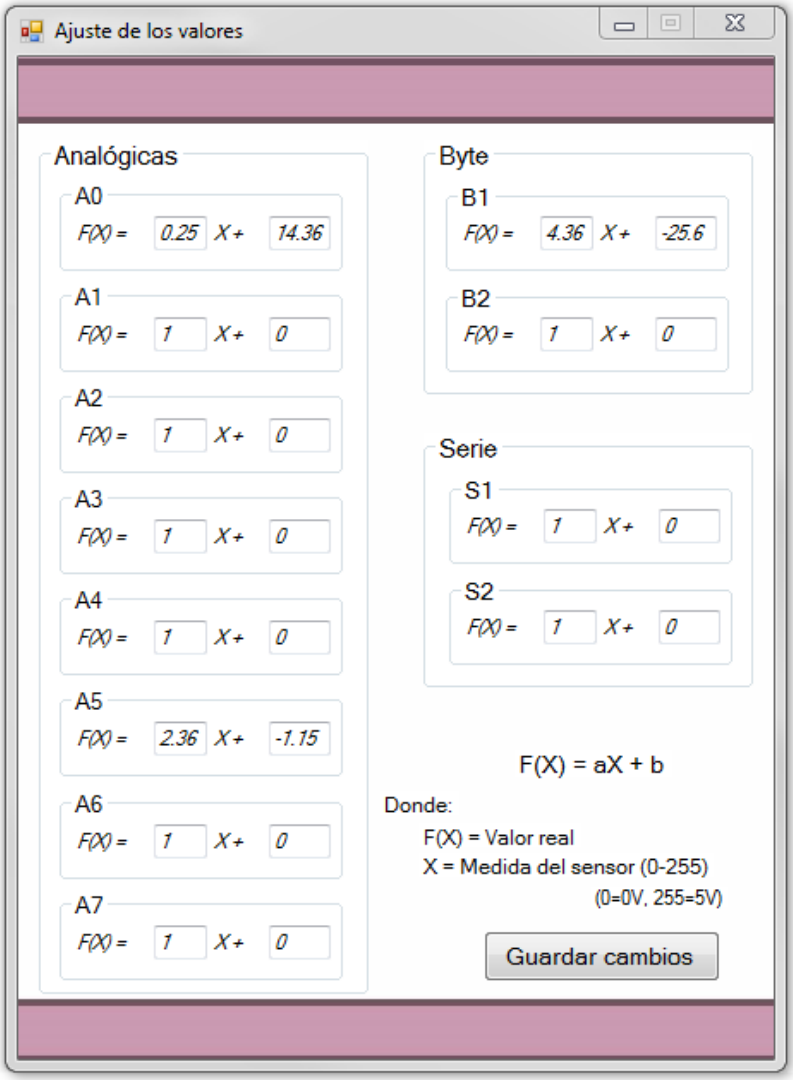


Ilustración 27 – Ventana de ajuste de valores

- ✓ En el campo "Puerto COM" se deberá introducir el puerto a través del cual se conectará la placa correspondiente al nodo maestro de la red con el PC.

- ✓ Una vez configurada la red, se podrá proceder a la puesta en marcha del sistema pulsando el botón "Comenzar". Tras esto aparecerá la etiqueta "Sistema conectado" y no se permitirá cambiar la configuración mientras esté conectado. Para poder cambiar la configuración, habrá que pulsar el botón "Detener" y se desbloquearan los campos que permitan modificar el sistema.

- ✓ Al pulsar el botón "Ayuda" situado en la parte inferior izquierda, aparecerá una ventana con las instrucciones a seguir para realizar la configuración del sistema.

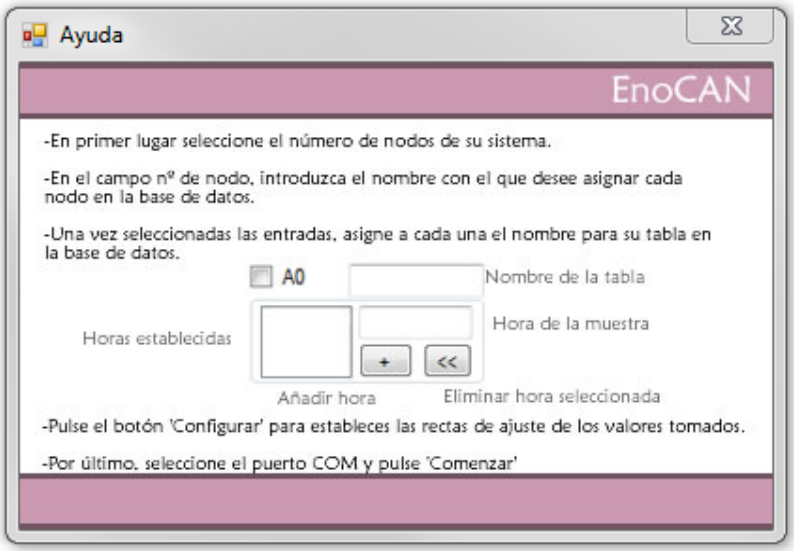


Ilustración 28 - Ventana ayuda

✓ Al pulsar el botón "Visualizar" situado en la parte inferior derecha, se podrá acceder a la ventana desde la que se pueden representar los datos de la base.

Representación de datos

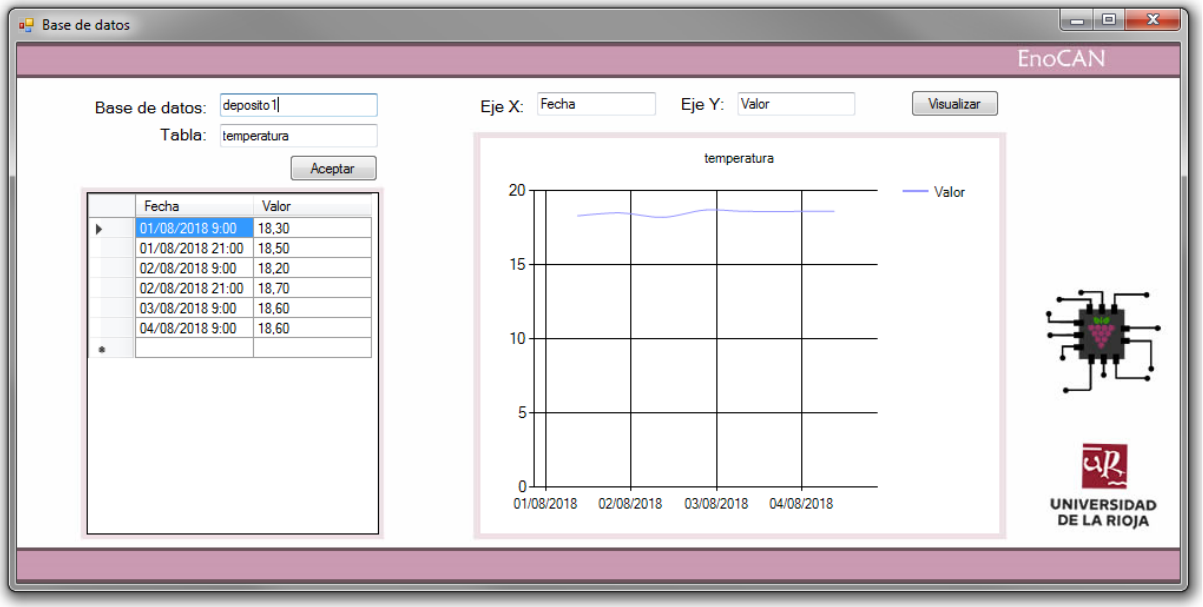


Ilustración 29 – Ventana de representación de valores de la base de datos

Desde esta ventana se puede acceder a los datos almacenados en la base de datos para su representación bien en forma de tabla o de gráfico. Para acceder a los mismos se debe introducir el nombre del nodo que se quiere consultar y la tabla perteneciente a ese nodo. Una vez representados los datos en forma de tabla, se puede proceder a su representación gráfica. Para ello se debe seleccionar qué columna de la tabla se representa en cada eje.

2.9.4 Red de nodos vía CAN bus

A la hora de desarrollar el firmware que correrá en las placas, uno de los requisitos de diseño del proyecto es que sea un firmware común para todas ellas (Ver apartado "*2.6.1 Requisitos de diseño de la red de nodos*"). Mediante este firmware las placas han de ser capaces de trabajar como nodo maestro como nodo esclavo. La dirección de cada nodo es lo que define el tipo de nodo, y se podrá asignar mediante un interrupotor switch externo en cada placa.

Estructura de las tramas

Se ha desarrollado un sistema de tramas específico para este proyecto. Para ello se han creado varios tipos de trama en función de quién transmita o reciba la información y el modo en que se transmite.

Ilustración 30 - Flujo de la información del sistema

Para la comunicación de la aplicación con la base de datos no ha sido necesario crear una trama específica. Se ha realizado a través del propio lenguaje MySQL utilizando las distintas funciones que ofrece la extensión de esta base de datos para Visual Basic.

Cada campo de las tramas corresponde a un bloque de 1 byte. Según el contenido del campo puede ser un valor comprendido entre 0-255 o bien un carácter en formato ASCII.

Aplicación Visual Basic → Nodo Maestro.

La comunicación de la aplicación con la red de nodos se realiza a través del puerto serie del PC con el nodo maestro. El formato de la trama es el siguiente:

Dirección del nodo	Tipo de entrada	Nº de entrada	CS	#

Tabla 6 – Formato de trama Visual Basic - Nodo Maestro

- Dirección del nodo: indica el nodo al que va dirigida la trama. La dirección del nodo maestro será la 0x100. El contenido de este campo es el offset que hay que sumarle a la dirección del nodo maestro para obtener la del nodo esclavo al que va dirigido.

- Tipo de entrada: Canal al que pertenece la entrada. Indica si se trata de una entrada Analógica ("A"), digital ("D"), bloque de 1 byte ("B") o tipo serial ("S").

- Nº de entrada: Acompañado del tipo de entrada, indica el pin de la placa al que corresponde la entrada.

- CS: se envía un checksum a modo de byte de comprobación para que el receptor pueda detectar si ha habido algún cambio en el mensaje durante su transmisión. Es el producto de realizar la operación XOR todos los campos de la trama, incluido el carácter fin de

trama. El nodo maestro comprobará que la trama está íntegra si al realizar la XOR de todos los campos incluido el CS da cero como resultado.

Un ejemplo de trama de este tipo sería el siguiente:

(	A	0	Y

Tabla 7 - Ejemplo de trama Visual Basic - Nodo Maestro

- "(": este carácter corresponde a un valor de 40d (28h). La dirección del nodo al que va dirigido la trama será la 0x128.
- "A0": La entrada de la que se desea la muestra es la Analógica 0.
- "Y: Valor del CS de la trama. (⊕ A ⊕ 0 = Y

Nodo Maestro → Nodo Esclavo

Una vez recibe el nodo maestro las instrucciones por parte de la aplicación de qué nodo y entrada se desea una muestra, el nodo maestro envía mediante CAN bus la petición al nodo correspondiente. Una trama CAN tiene la siguiente forma:

Campo de inicio	Campo de arbitrio	Campo de control	Campo de datos	Campo de aseguramiento (CRC)	Campo de confirmación (AKC)	Campo de fin de trama (EOF)

Tabla 8 - Formato de trama Nodo Maestro- Nodo Esclavo

En el apartado "*3.2 Anexos CAN bus*" e explica con detalle el contenido y función de cada campo. El campo de datos es el que contiene la información a transmitir. Este campo tiene una longitud máxima de 8 registros de 1 byte (la longitud del mensaje se define en el campo de control). En este caso únicamente se necesitarán 2 bytes, ya que el nodo maestro únicamente envía al esclavo el tipo de entrada y el número de entrada que desea leer codificados en ASCII. El campo de datos de esta trama tiene la siguiente forma:

Campo de datos	
Dato[0]	Dato[1]
Tipo de entrada	Nº de entrada

Campo de datos	
Dato[0]	Dato[1]
"D"	"2"

Tabla 9 - Campo de datos de la trama CAN

Nodo Esclavo → Nodo Maestro

Tras la petición realizada por parte del nodo maestro para la toma de una muestra, el nodo esclavo enviará al nodo maestro la respuesta a esa petición. En este caso se van a necesitar dos bytes más en el campo de datos, que corresponderán al valor de la muestra y al nodo de la que procede. Es importante que el nodo esclavo informe al nodo maestro sobre quién envía el mensaje, ya que cada nodo esclavo solo recibe mensajes del nodo maestro; pero el nodo maestro recibe mensajes de todos los nodos esclavo.

Campo de datos			
Dato[0]	Dato[1]	Dato[2]	Dato[3]
Dirección del nodo	Tipo de entrada	Nº de entrada	Valor de la muestra

Campo de datos			
Dato[0]	Dato[1]	Dato[2]	Dato[3]
40	"B"	"1"	252

Tabla 10 - Formato de trama Nodo Esclavo - Nodo Maestro

Nodo Maestro → Aplicación Visual Basic

El último paso del proceso de solicitud-recogida de daos, es la transmisión desde el nodo maestro a la aplicación de la información solicitada. La transmisión se realiza de nuevo mediante puerto serie. El mensaje tiene el siguiente formato:

Dir. del nodo procedente	;	Tipo de entrada	;	Nº de entrada	;	Valor	;	CS	;	#

Tabla 11 - Formato de trama Nodo Maestro - Visual Basic

- Dirección del nodo procedente: indica el nodo del que procede la muestra.
- Tipo de entrada: Canal al que pertenece la muestra. Indica si se trata de valor analógico, digital, etc.
- Nº de entrada: indica el pin de la placa al que corresponde la muestra.
- CS: checksum de comprobación. La aplicación comprobará que la trama está íntegra si al realizar la XOR de todos los campos incluido el CS, el carácter de final de trama y los caracteres de separación, da cero como resultado.
- ";" : carácter de separación de campos del mensaje.
- "#": carácter de fin de trama. Debido al procedimiento que sigue la aplicación a la hora de recibir información vía serie, es necesario delimitar las tramas con este carácter para que se pueda detectar cuando acaba un mensaje recibido y empieza otro.

Si el nodo maestro detecta mediante el checksum que el mensaje recibido desde la aplicación no es íntegro, le enviará la trama de error ***"1;#".***

Diagramas del firmware

Los programas desarrollados en el entorno Arduino, y por tanto en el entorno MPIDE, se dividen en tres bloques:

- Declaraciones: se declaran e inicializan las constantes, variables, funciones y librerías utilizadas.
- Set Up: esta función solo se ejecuta una vez al arrancar el programa.
- Loop: esta función contiene el código que se ejecutará continuamente. Es el núcleo del programa, y se usa para el control activo de la placa. Se ejecuta justo después del Set up.

Según se identifique la placa como nodo maestro o como nodo esclavo, se realizarán unas funciones u otras. Las acciones del nodo maestro son:

- Recibir la trama por parte de la aplicación del PC vía serie, donde se indica el nodo y entrada que se quieren leer.
- Enviar vía CAN bus la solicitud de lectura de la entrada al nodo esclavo indicado.
- Recibir vía CAN bus la información por parte del nodo esclavo de la lectura de la entrada solicitada.
- Enviar vía serie al PC los datos del nodo y entrada que se ha solicitado leer.

Las acciones de los nodos esclavo son:

- Recibir vía CAN bus la solicitud de lectura de una entrada.
- Proceder a la lectura de la entrada indicada.
- Enviar vía CAN bus al nodo maestro la información de la lectura de la entrada solicitada.

A continuación, se van a representar los diagramas de flujo de las funciones principales del firmware y como interactúan entre sí. El desarrollo del código completo comentado del firmware se encuentra en el apartado "*3.5 Código del firmware*",

Set Up y Loop

La primera acción que realiza el programa es la lectura del interruptor switch externo para asignar la dirección del nodo asociado a la placa. Una vez que se ha asignado una dirección a la placa, se inicializa el nodo para esa dirección. Ya inicializado el sistema, la placa comenzará a operar como maestro o como esclavo en función de la dirección que se le haya asignado. Operará como nodo maestro si su dirección es la 0x100 y como nodo esclavo si es otra.

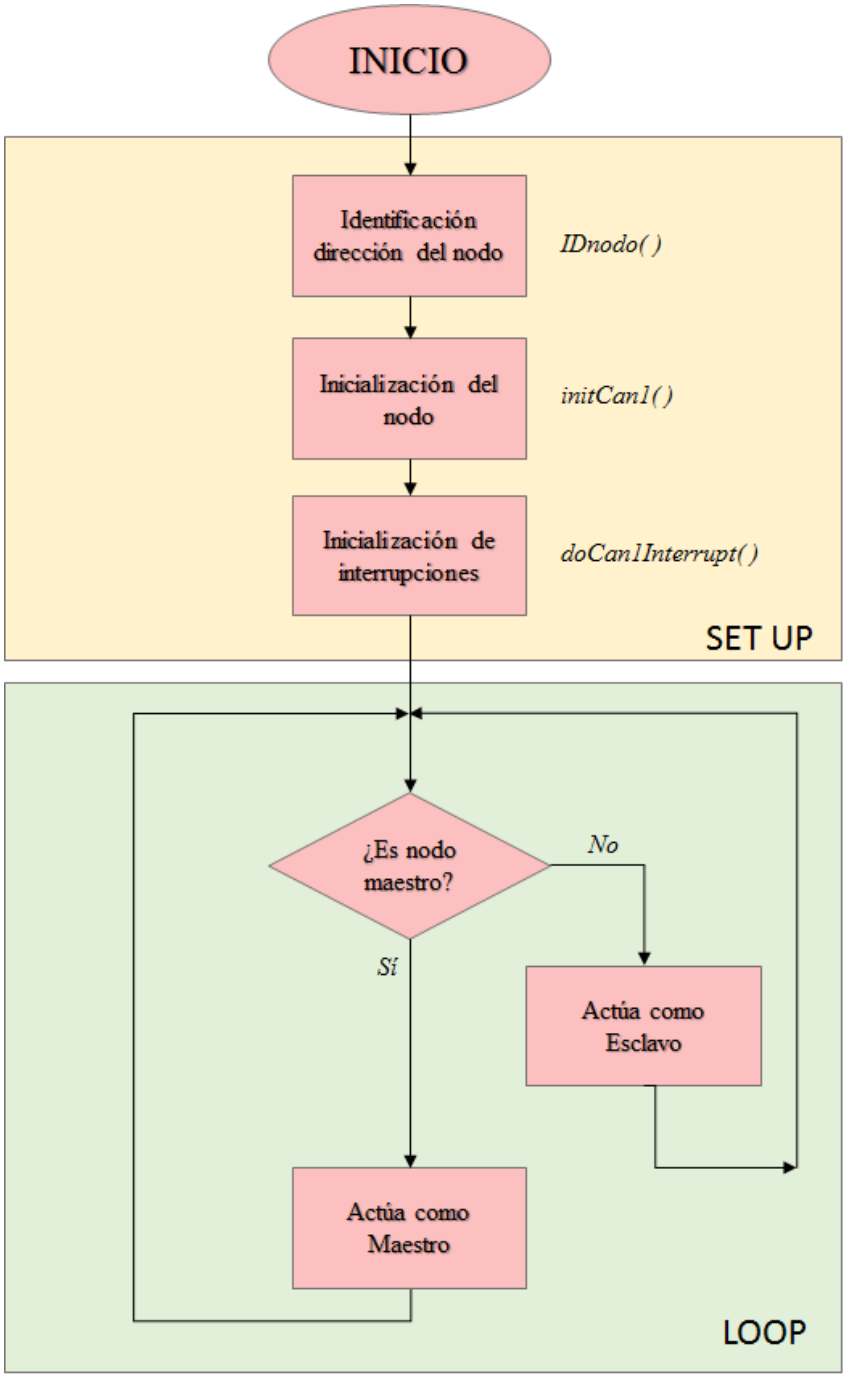


Ilustración 31- Diagrama de flujo del firmware

initCan1(nodo)

Esta función es la encargada de inicializar y configurar la comunicación CAN. Tiene como entrada la dirección del nodo asignado a la placa.

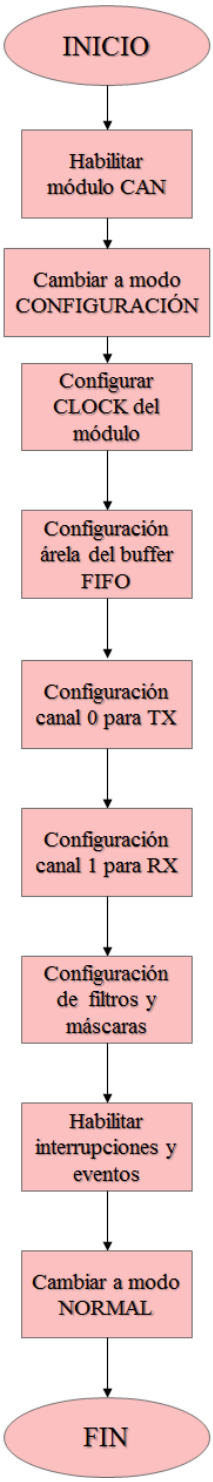


Ilustración 32 - Diagrama de flujo de la función iniCan1

DoCan1Interrupt ( )

Esta función se ejecuta cuando se recibe un dato. Su misión es habilitar la lectura de los datos recibidos.

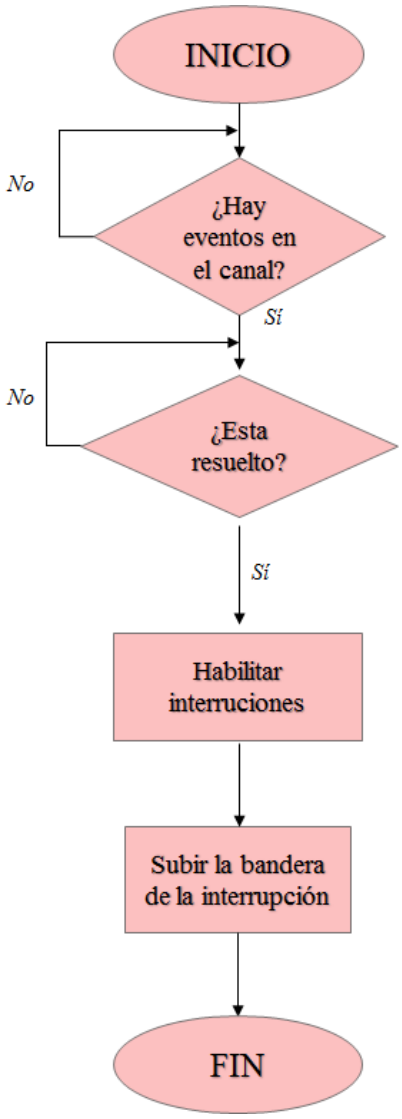


Ilustración 33 - Diagrama de flujo de la función doCan1Interrupt

Loop() como maestro

Cuando la placa funciona como maestro, realiza en bucle el siguiente procedimiento. En primer lugar, comprueba si se ha recibido algún mensaje por el puerto serie. Si hay alguna solicitud pendiente a un nodo esclavo, se identifica a qué nodo va dirigida y se envía por el bus CAN a ese nodo la solicitud de toma de muestra. Si no hay ninguna solicitud pendiente, simplemente comprueba el puerto serie y la recepción CAN a la espera de mensajes.

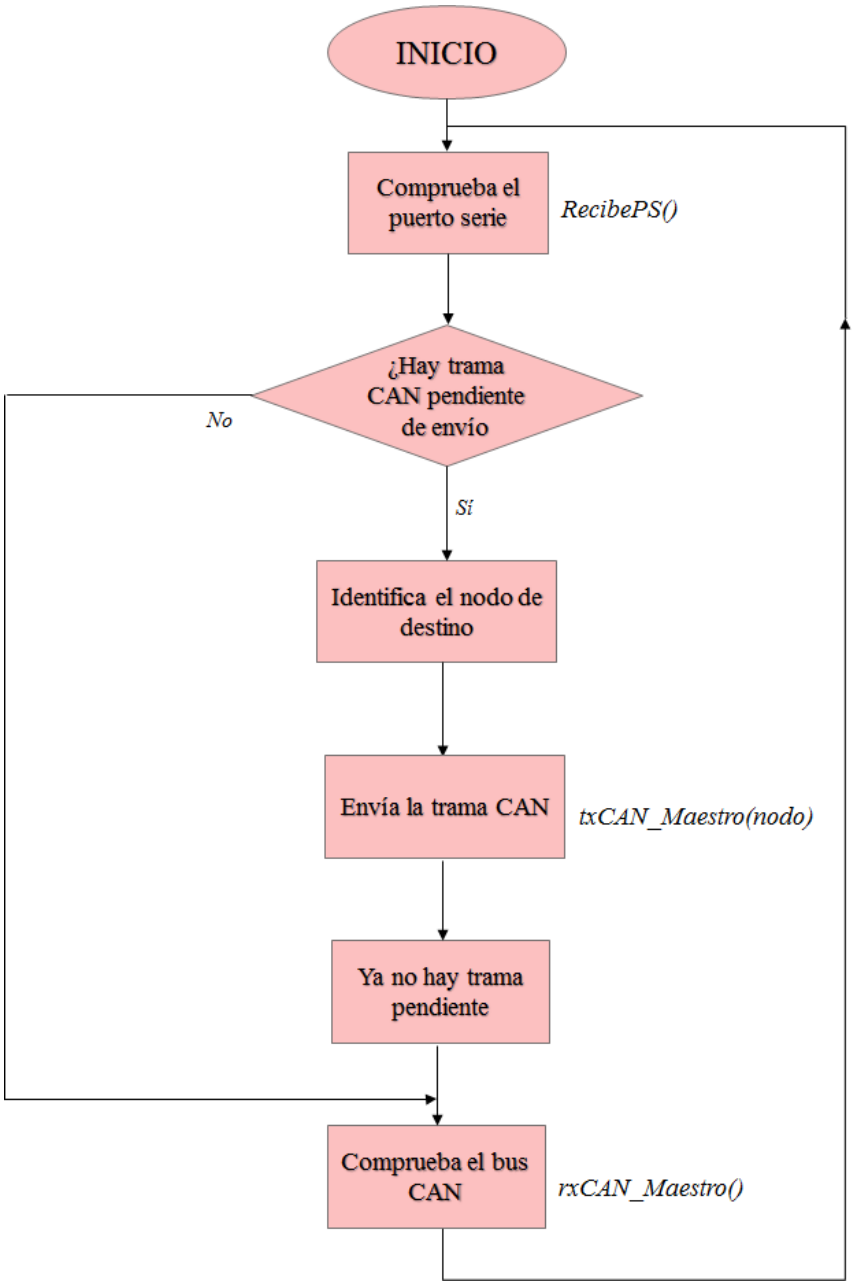


Ilustración 34 - Diagrama de flujo de la función loop operado como Maestro

RecibePS()

A través de esta función se recibe la solicitud de lectura de datos por parte de la aplicación, y se comprueba la integridad del mensaje a través del checksum. Si la trama no presenta ningún error, se solicita el envío al nodo esclavo.

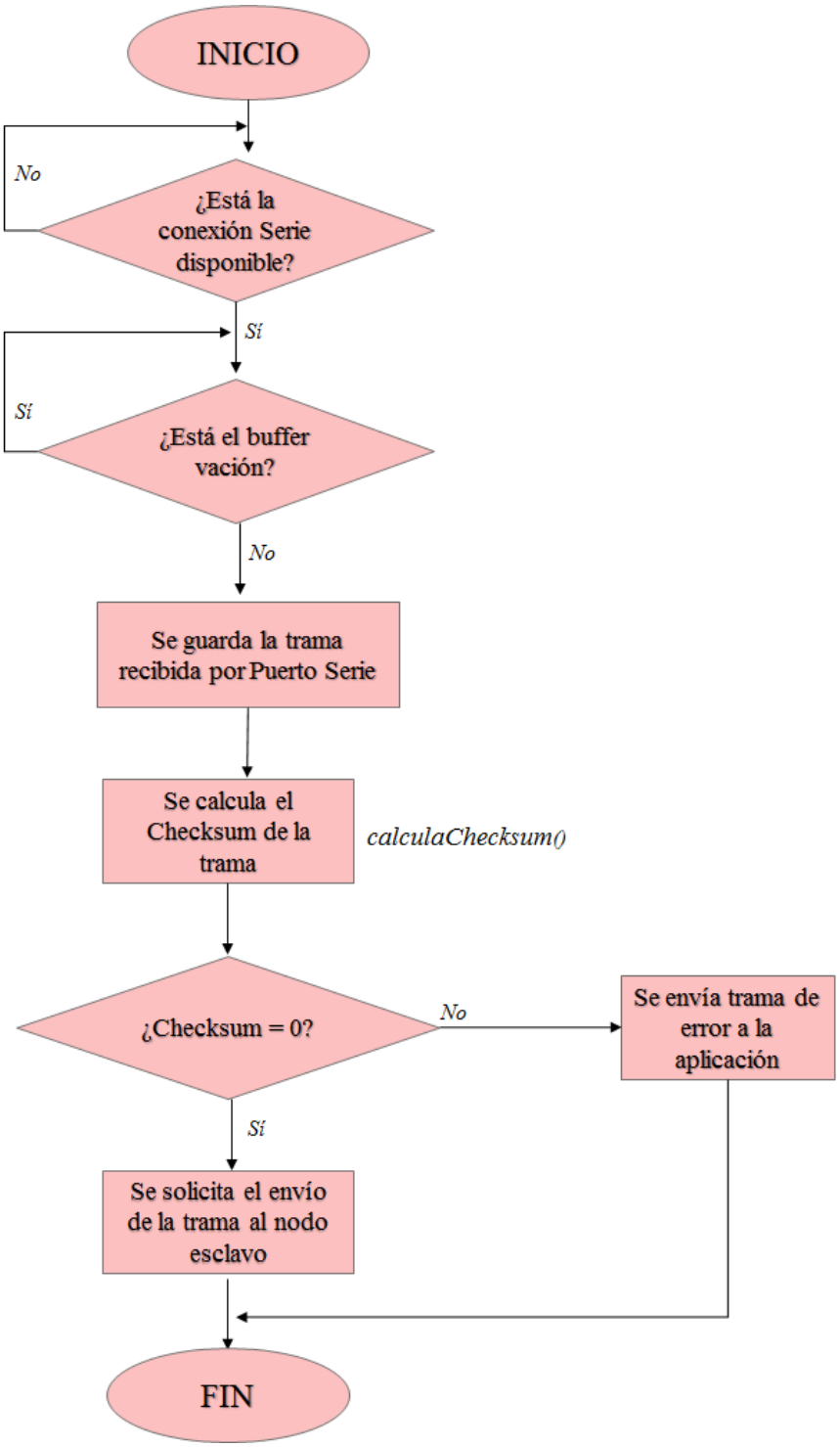


Ilustración 35 - Diagrama de flujo de la función recibePS

TxCAN_Maestro(nodo)

Mediante esta función, el nodo maestro le envía al esclavo la información de la entrada que se desea leer.

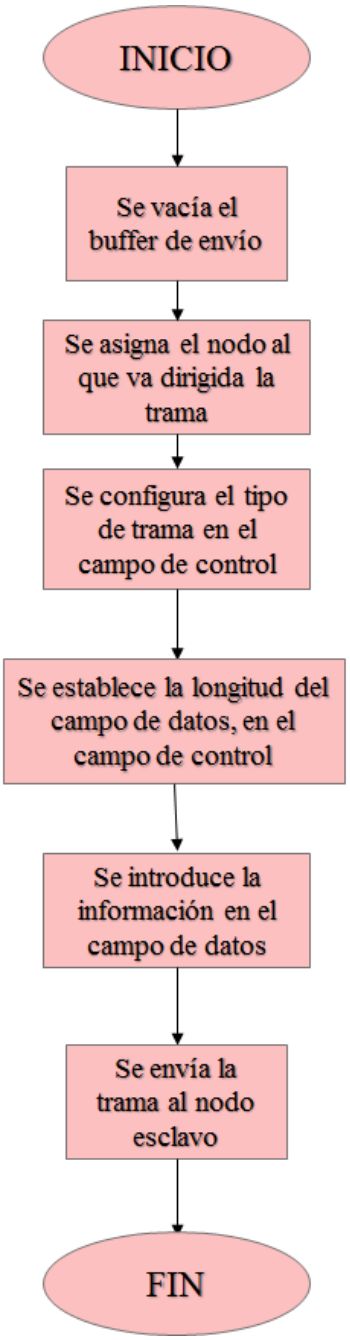


Ilustración 36 - Diagrama de flujo de la función txCAN_Maestro

RxCAN_Maestro()

A través de esta función, se recibe la información sobre la lectura de la entrada del nodo esclavo. Con esa información se crea la trama de envió de información al PC a través del puerto serie y se procede a su envío.

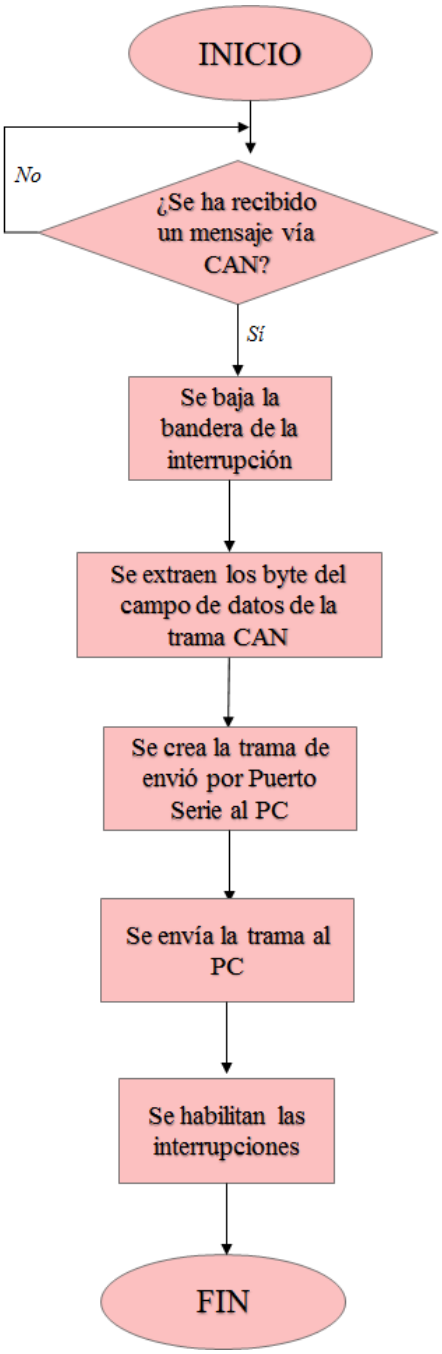


Ilustración 37 - Diagrama de flujo de la función rxCAN_Maestro

Loop() como esclavo (rxCAN_Esclavo())

Cuando la placa funciona como maestro, realiza en bucle la función habilita la recepción vía CAN. La realiza mediante la función *rxCAN_Esclavo( ).* Todas las demás acciones derivan de esta función, ya que, si no recibe ninguna solicitud para tomar una muestra, no se realiza ninguna acción.

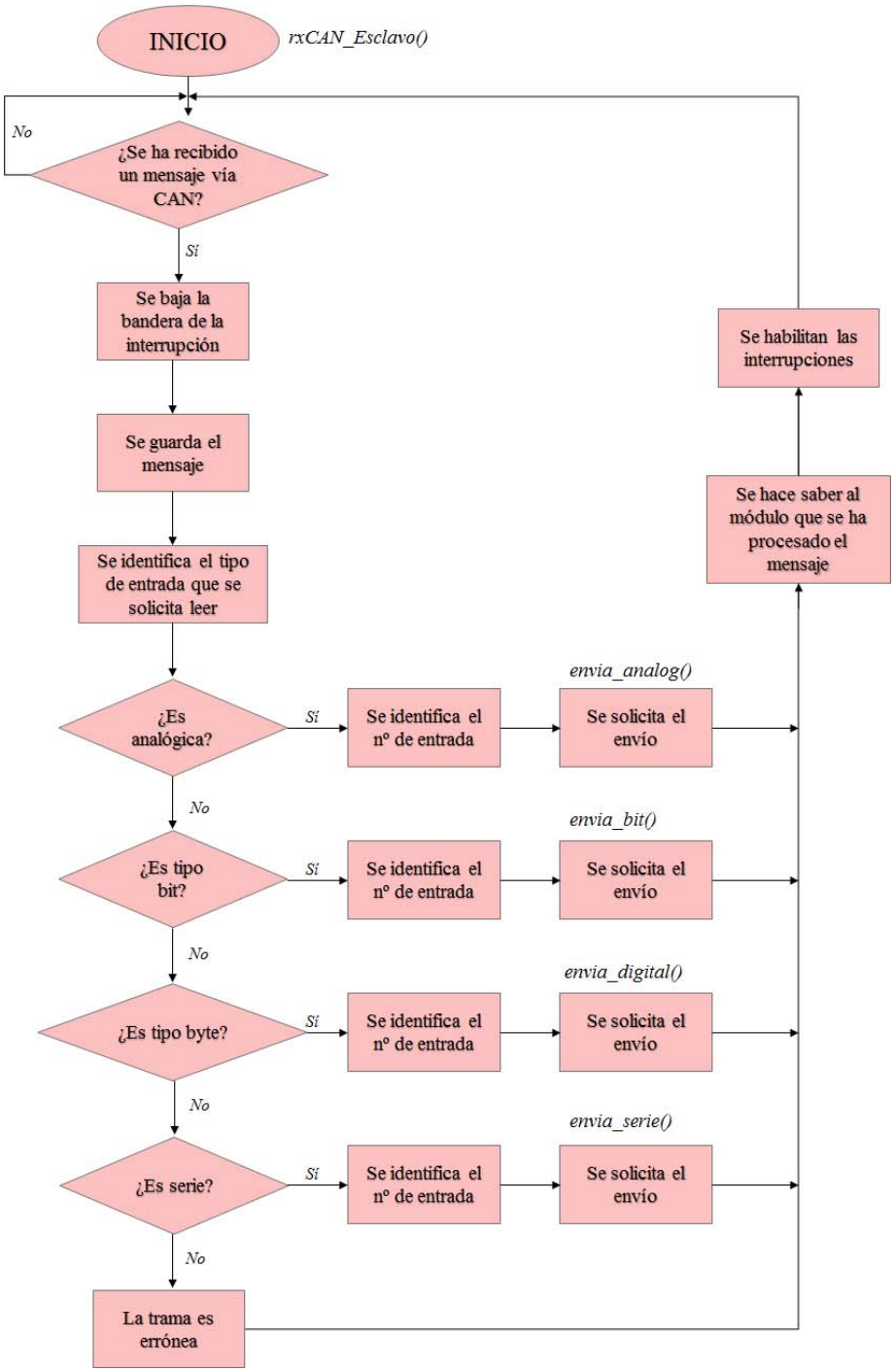


Ilustración 38 - Diagrama de flujo de la función loop como esclavo (función rxCAN_Esclavo)

Envia_analog/digital/bit/serie(nodo, nº entrada)

A pesar que de que se han creado 4 funciones, estas funciones realizan las mismas acciones, pero adecuadas al tipo de entrada que se desea leer. Se habilita y el pin de entrada y se procede a su lectura. Una vez obtenido el valor de la entrada se guarda los datos de dirección de nodo, tipo de entrada, número de entrada y valor, para su envío a través de la red CAN al nodo maestro.

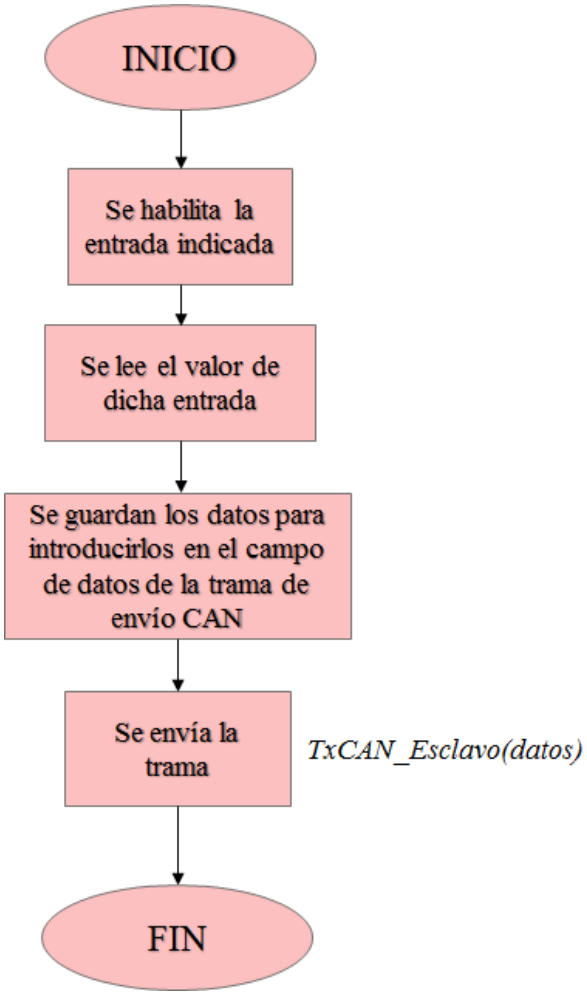


Ilustración 39 - Diagrama de flujo de la función envia_analog/digital/bit/Serie

TxCAN_Esclavo(datos)

Mediante esta función el nodo esclavo envía los datos de la lectura de las entradas al nodo maestro vía CAN bus.

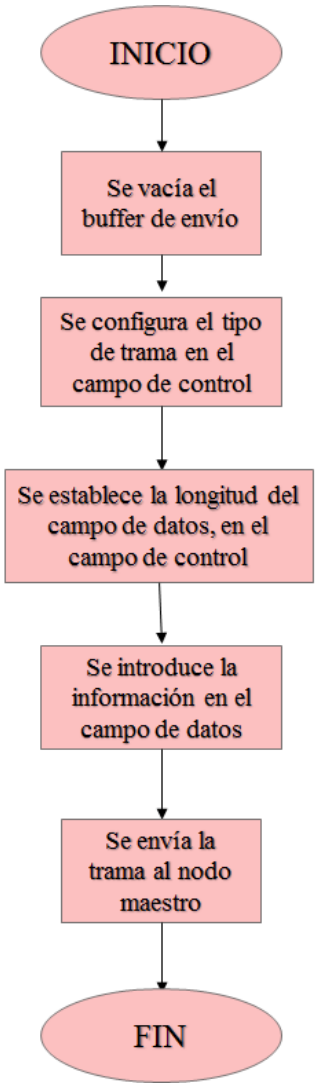


Ilustración 40 - Diagrama de flujo de la función txCAN_Esclavo

2.9.5 Conexión de un nodo al sistema

MCP2551

Los elementos que forman una red can bus son el bus, las resistencias de cierre, controlador y transceiver (ver apartado "*3.2 Anexos del bus CAN*"). Cada nodo del sistema debe tener su propio controlador y transceiver. En el caso del controlador, la placa base ChipKIT Max 32 cuenta con dos de ellos ya integrados, lo que permite comunicar el microprocesador con el transceiver acondicionando la información que entra y sale entre ambos componentes.

El transceiver es el elemento a través el cual se envían y reciben los mensajes. Su misión es convertir la secuencia de datos procedentes del controlador CAN a los niveles del bus, así como acondicionar y preparar la información procedente del bus para que pueda ser utilizada por los controladores. Para ello, convierte la señal procedente de los terminales Tx y Rx de la placa en la señal CAN H y CAN L del bus. Además, amplifica la señal cuando la información se vuelca en la línea y la reduce para suministrarla al controlador. En este proyecto, se va a utilizar el transceiver MCP2551.

Ilustración 41- Circuito integrado MCP 2551 para CAN bus

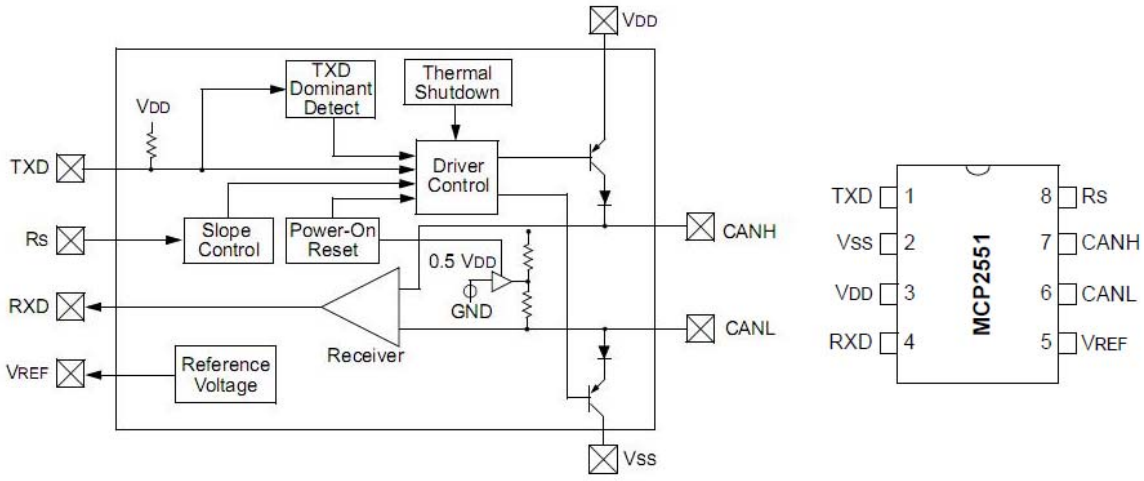


Ilustración 42 - Diagrama de bloques del chip MCP 2551

El modo de operación de este chip se puede configurar a través de la tensión de entrada del pin RS. Puede trabajar en tres modos: stand by, alta velocidad y slope-control.

Modo	Corriente en el pin RS	Tensión resultante en el pin RS
Stand by	-10 IRS < 10 A	VRS > 0,7 VDD
Slope-Control	10 A < -IRS < 200 A	0,4 VDD < VRS < 0,6 VDD
Alta velocidad	-IRS < 610 A	0 < VRS < 0,3V DD

Tabla 12 - Modos de operación del chip MCP2551

- Stand by: el transmisor está apagado y el receptor trabaja con una corriente menor por lo que opera a una velocidad menor.

- Slope-control: se limitan los tiempos de subida y bajada de CAN H y CAN L. La pendiente del flanco se controla conectando una resistencia entre el pin RS y tierra. La pendiente será proporcional a la corriente de salida en este pin.

- Alta velocidad: en este modo los tiempos de subida y de baja del transmisor son elevados, lo que permite operar a una elevada velocidad. Éste será el modo en el que trabajará el transceiver en el sistema de este proyecto.

Identificación y direccionamiento del nodo

El protocolo can está orientado a mensajes. Esto quiere decir que el mensaje no va dirigido a ningún nodo o unidad de mando en concreto, sino que todos los nodos reciben el mensaje. Este mensaje tiene un identificado el cual permite a cada nodo reconocer si el mensaje le interesa o no, y en consecuencia, aceptarlo.

Cuando un nodo recibe un mensaje, éste realiza un filtrado para determinar si procesa el mensaje o no. El campo de la trama encargado de filtrar el mensaje es el campo de arbitrio, que contiene el identificador del mensaje.

Campo de inicio	Campo de arbitrio	Campo de control	Campo de datos	Campo de aseguramiento (CRC)	Campo de confirmación (AKC)	Campo de fin de trama (EOF)

Tabla 13 - Campos de una trama CAN

En este campo permite configurar los filtros y máscaras para determinar si el mensaje se debe aceptar y procesar. A través de las máscaras se informa al controlador qué bits del identificador han de ser ignorados por los filtros; y a través de los filtros se indica al controlador qué valor ha de tener el identificador. Para cada bit, la tabla de verdad resultante es la siguiente:

Bit de la máscara	Bit del filtro	Bit del identificador	Resultado
0	x	x	Aceptar
1	0	0	Aceptar
	1	1	Aceptar
	0	1	Rechazar
	1	0	Rechazar

Tabla 14 - Tabla de verdad del filtrado de una trama CAN

Un requisito de este proyecto es la utilización de un firmware único para el control de las placas. Por ello es necesario que la asignación de la dirección de cada nodo se realice de forma externa. Para este cometido se ha optado por la utilización de un DIP switch, conectado a la placa, de tal forma que al arrancar el programa se procede a la lectura de este switch para asignar el identificador del nodo.

Al tratarse de un prototipo, el sistema desarrollado para este proyecto únicamente permite la configuración de 10 nodos, a parte del nodo maestro (nodo 0). Para ello bastará con utilizar un switch de 4 entradas como el que se muestra en la siguiente ilustración.

Ilustración 43 - DIP Switch de 4 posiciones

Valor del switch	0	1	2	3	4	5	6	7	8	9	10
Dirección	0x100	0x128	0x129	0x12A	0x12B	0x12C	0x12D	0x12E	0x12F	0x130	0x131

Tabla 15 - Direcciones de los nodos del sistema

Cuando se realice un envío a un nodo, el valor del filtro corresponderá con la dirección del nodo de destino. Tal y como muestra la "*Tabla 15 – Tabla de verdad del filtrado de una trama CAN*", para que el filtro identifique la dirección completa del nodo, la máscara ha de tener un valor de 0xFFF. La configuración de filtros y máscaras se puede observar en el apartado "*3.4 Código del firmware*"

Conexión de los elementos de un nodo

Como el alcance de este proyecto es el desarrollo de un prototipo a nivel de laboratorio, se va a realizar la conexión de este chip mediante una placa de inserción. A continuación, se muestra el montaje de un nodo:

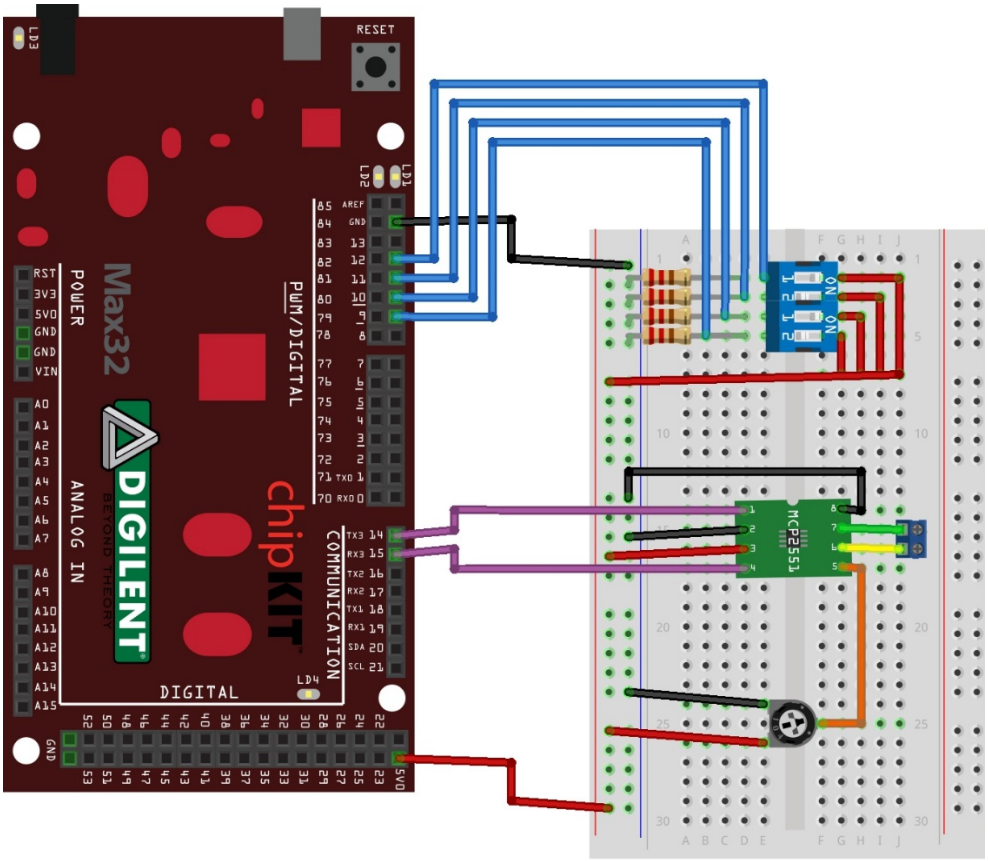


Ilustración 44 - Conexionado en protoboard de un nodo del sistema

CHIPKIT MAX 32	
PIN	***Descripción***
9	Bit 0 de la dirección del nodo
10	Bit 1 de la dirección del nodo
11	Bit 2 de la dirección del nodo
12	Bit 3 de la dirección del nodo
14	TX
15	RX

MCP2551	
PIN	***Descripción***
1	TX
2	GND
3	VCC
4	RX
5	VSS
6	CAN L
7	CAN H
8	RS

Tabla 16 - Asignación de pines de cada nodo

2.9.6 Funcionamiento del sistema

En este apartado se va a mostrar el funcionamiento del prototipo a través de la configuración de un nodo maestro y un nodo esclavo. Como todos los nodos presentan el mismo firmware, es posible variar el número de nodos sin que afecte a la red. Por ello, una vez resuelto el funcionamiento con un nodo esclavo, se puede extrapolar a un sistema con un mayor número de nodos únicamente conectando los nuevos nodos a la red y configurándolos desde la aplicación.

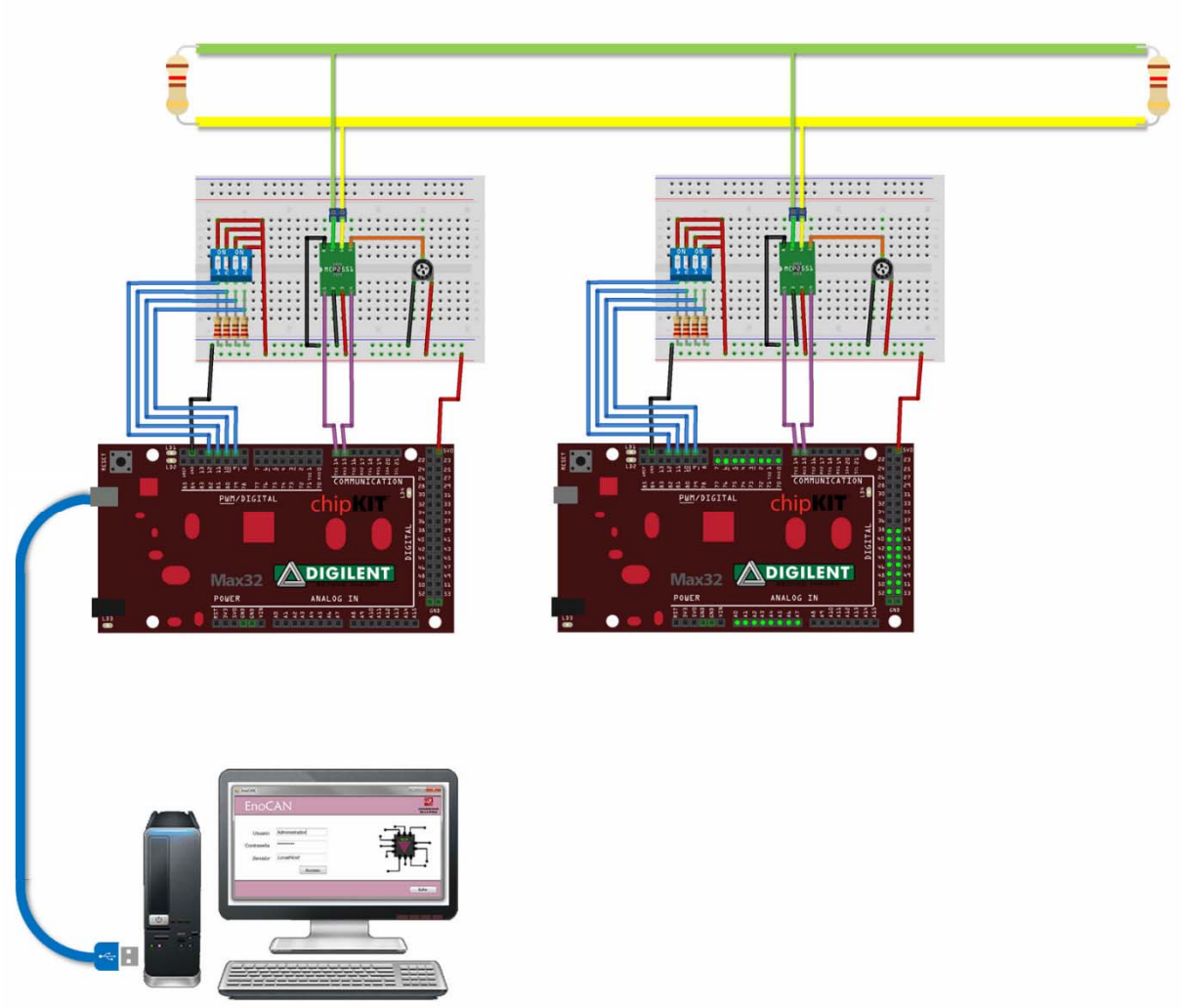


Ilustración 45 - Conexión de un sistema con un nodo esclavo

Se va a partir de una configuración de la aplicación en modo buffer para observar cómo se almacenan los datos en un archivo del pc antes de su inserción en la base de datos. Una vez obtenidas varias muestras, se va a conectar el sistema a la base de datos para que se introduzcan automáticamente los valores guardados en el buffer, y se va a proceder a su visualización en la aplicación.

En primer lugar, hay que acceder al sistema desde el modo buffer. Para ello, tras arrancar la aplicación, se introduce el usuario y contraseña asignados a este modo de operación.

- Usuario*: modobuffer*
- Contraseña: *passmodobuffer*

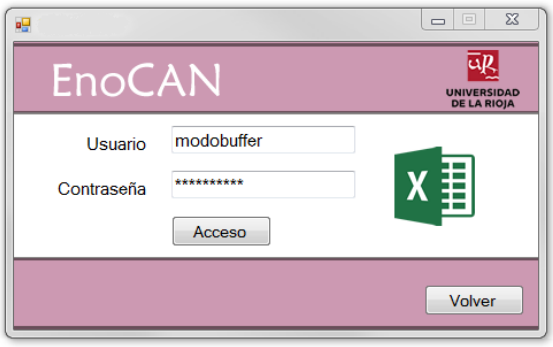


Ilustración 46 - Acceso al sistema desde el modo buffer

A continuación, tras seleccionar el número de nodos, se configuran las entradas que se van a emplear asignándoles un nombre para crear su tabla correspondiente en la base. En este caso se va a suponer que las señales proceden de dos sensores con salida 0-5V y un sensor con salida digital de 1 byte. Además, se seleccionan las horas a las que se desea que se lean los valores.

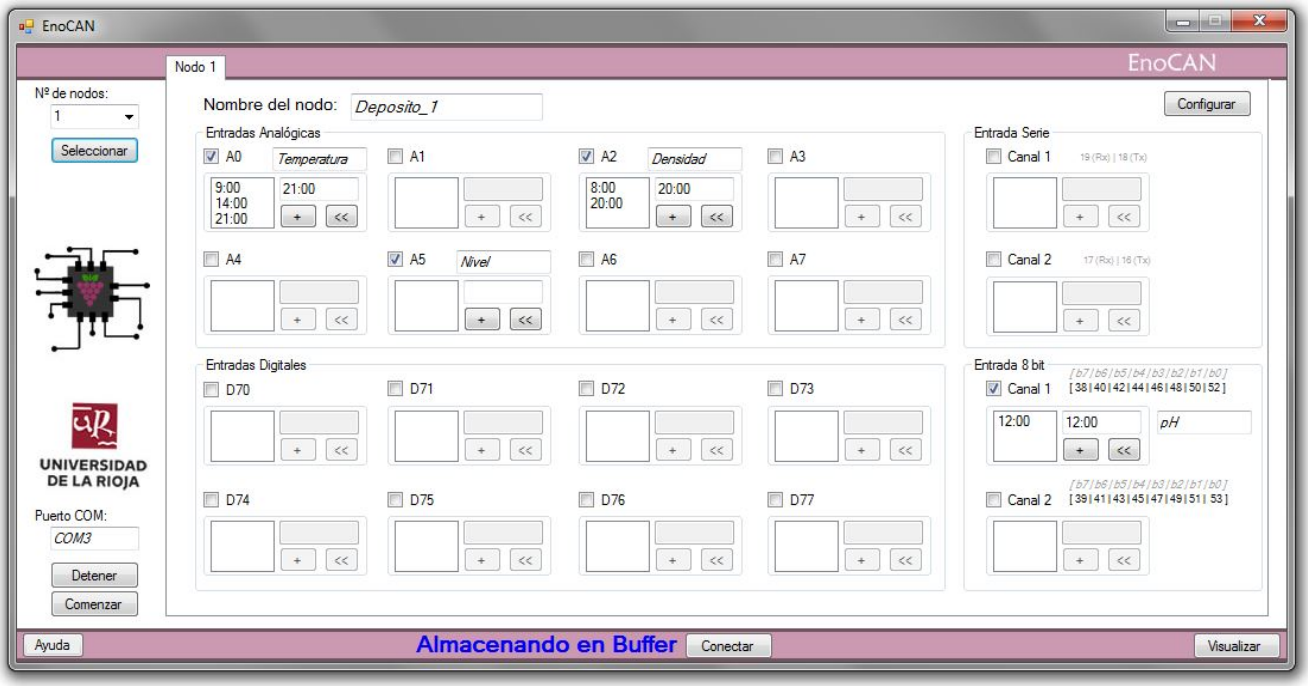


Ilustración 47 - Configuración de las entradas del nodo

Antes de comenzar con la lectura de valores, se tienen que configurar las funciones de conversión de los valores leídos para su escalado a valores reales. Para ello hay que consultar las funciones de transferencia de cada sensor en sus hojas de especificaciones.

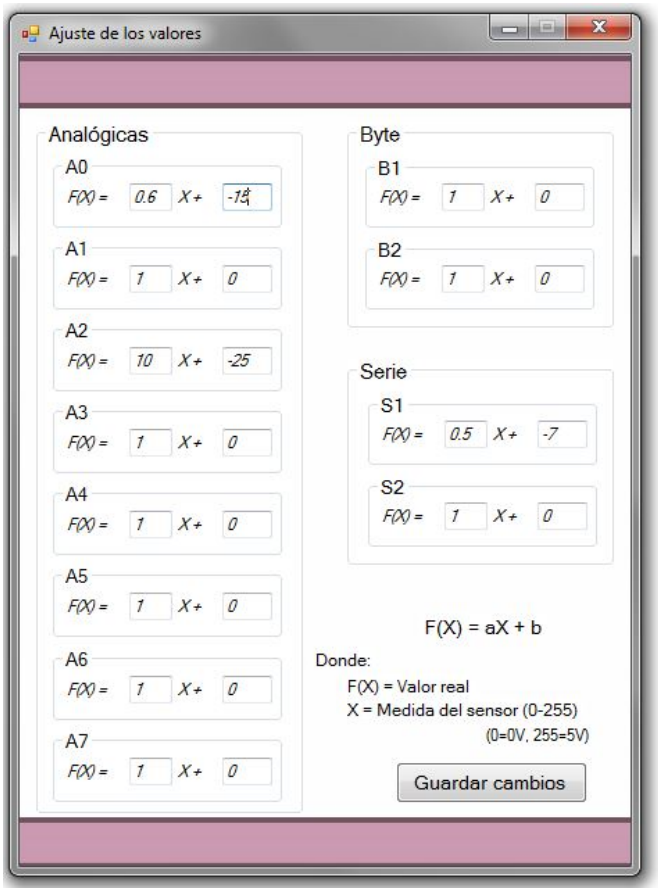
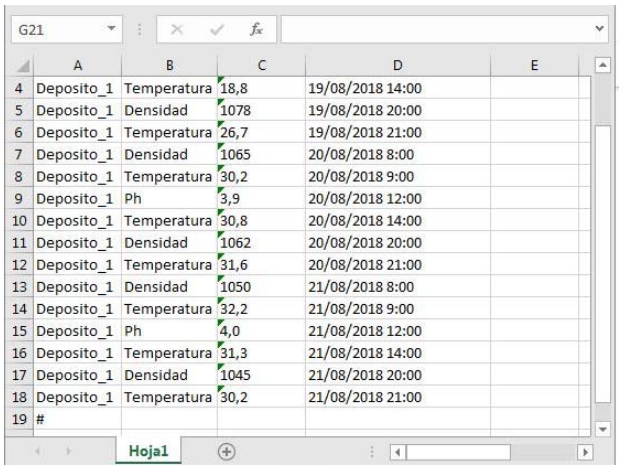


Ilustración 48 - Configuración de las funciones de transferencia

Una vez configuradas las entradas del nodo, se selecciona el puerto COM mediante el cual se va a realizar la comunicación con el nodo maestro y se establece la conexión. Como no se ha establecido conexión con la base de datos, los valores recibidos se van almacenando en el buffer. Este buffer es un archivo Excel que se crea automáticamente ubicado en el Escritorio. Tras unos días de funcionamiento del sistema, si se consulta este archivo se pueden observar los distintos valores que se han ido almacenando.



	A	B	C	D	E
4	Deposito_1	Temperatura	18,8	19/08/2018 14:00	
5	Deposito_1	Densidad	1078	19/08/2018 20:00	
6	Deposito_1	Temperatura	26,7	19/08/2018 21:00	
7	Deposito_1	Densidad	1065	20/08/2018 8:00	
8	Deposito_1	Temperatura	30,2	20/08/2018 9:00	
9	Deposito_1	Ph	3,9	20/08/2018 12:00	
10	Deposito_1	Temperatura	30,8	20/08/2018 14:00	
11	Deposito_1	Densidad	1062	20/08/2018 20:00	
12	Deposito_1	Temperatura	31,6	20/08/2018 21:00	
13	Deposito_1	Densidad	1050	21/08/2018 8:00	
14	Deposito_1	Temperatura	32,2	21/08/2018 9:00	
15	Deposito_1	Ph	4,0	21/08/2018 12:00	
16	Deposito_1	Temperatura	31,3	21/08/2018 14:00	
17	Deposito_1	Densidad	1045	21/08/2018 20:00	
18	Deposito_1	Temperatura	30,2	21/08/2018 21:00	
19	#				

Ilustración 49 - Contenido del buffer tras unos días de funcionamiento

Como se puede observar, se almacena el valor de la muestra y la fecha y hora en la que se realizó, así como del nodo del que procede. Los nombres del nodo y de la entrada corresponden con nombres de la base de datos y la tabla donde se introducirán los datos, tal y como se haya configurado desde la aplicación.

Para establecer conexión con la base de datos, se pulsa el botón 'Conectar' y se accede con un usuario registrado en la base de datos. Una vez conectado con el servidor, el sistema introduce los valores de las muestras a la base de datos según van entrando al buffer, introduciendo previamente los ya existentes.

Por último, se accede al formulario de visualización para poder acceder a las distintas tablas y representarlas gráficamente.

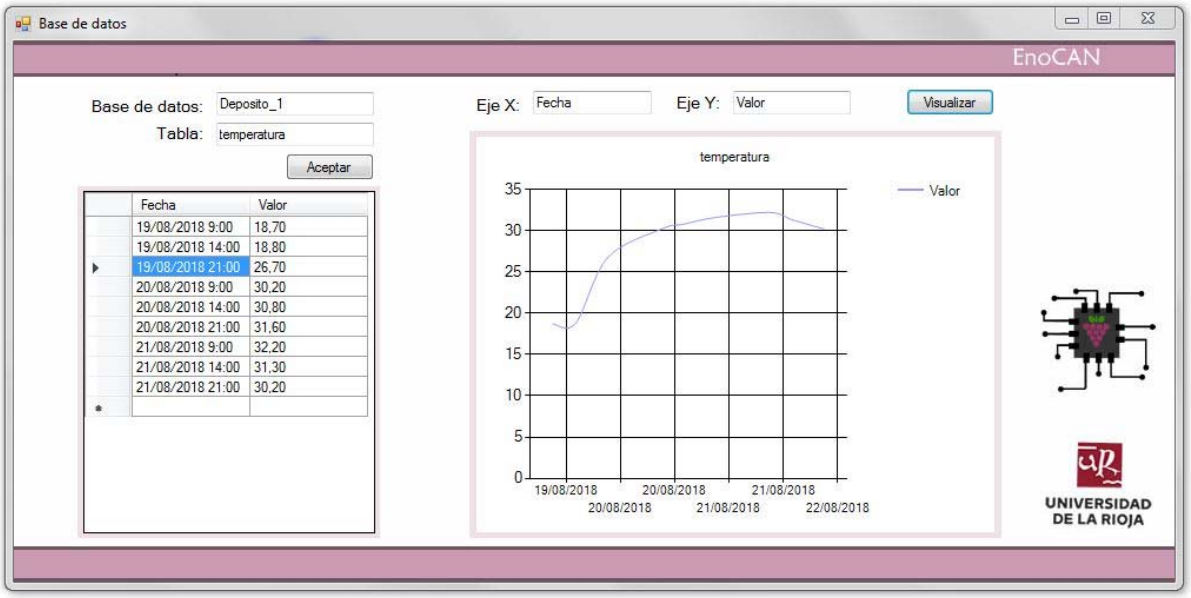


Ilustración 50 - Representación de la temperatura

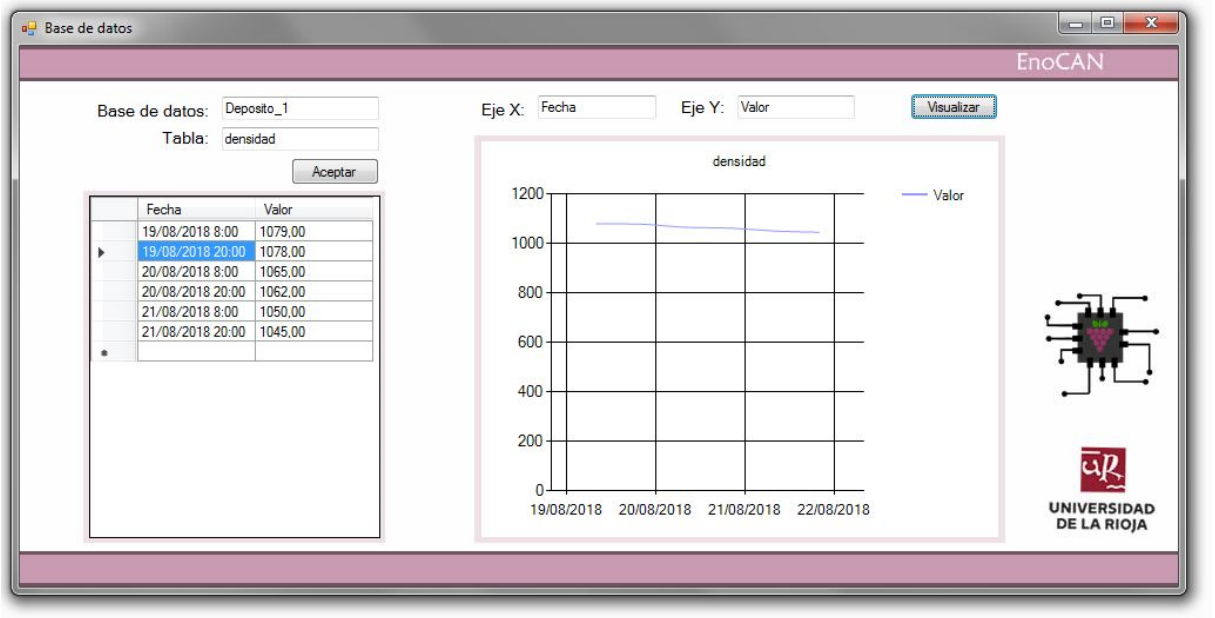


Ilustración 51 - Representación de la densidad

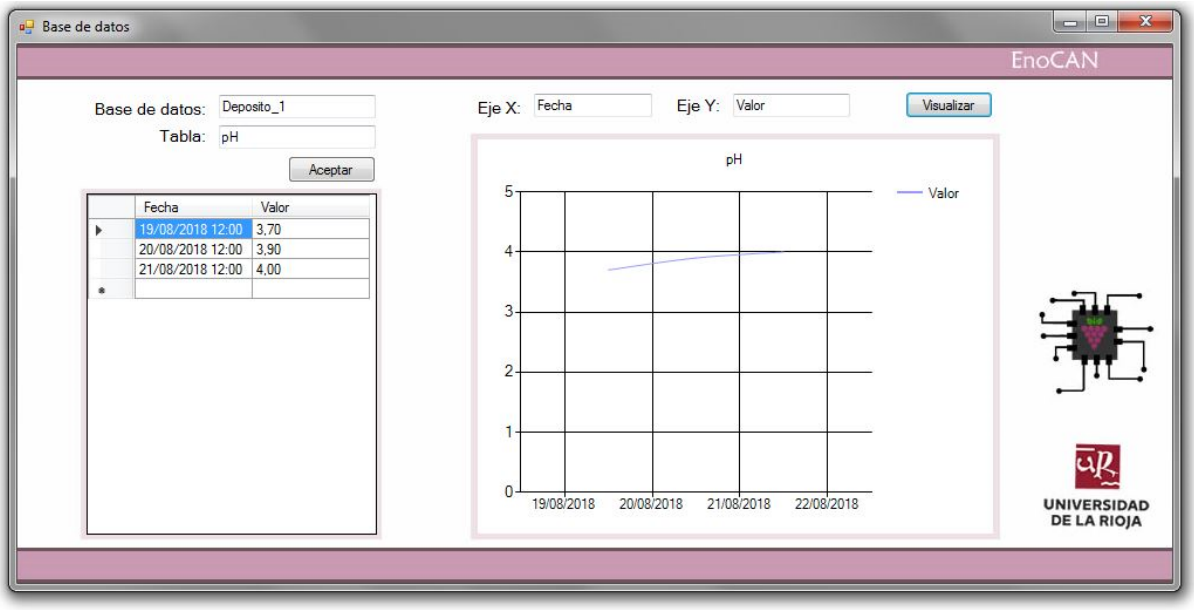


Ilustración 52 - Representación del pH

2.9.7 Conclusiones

En el presente proyecto se ha conseguido cumplir los objetivos generales y específicos previstos. El objetivo principal era el de diseñar un prototipo capaz de tomar y almacenar valores procedentes de distintos nodos conectados a una red.

Se ha desarrollado una aplicación software en VB.NET para gestionar y configurar la red de nodos, así como la base de datos donde se almacenará la información recogida. Esta aplicación está destinada a poder ser utilizada por varios usuarios registrados en la base de datos. Es capaz de configurar el número de nodos del sistema (con un máximo de 10 nodos) así como las entradas de cada nodo. Mediante esta aplicación se pueden representar los datos recogidos en una base de datos MySQL a través de tablas y gráficos.

Además, se ha logrado la interconexión de los nodos mediante una red de comunicaciones vía Can bus. Para el control de cada nodo de la red se han empleado placas ChipKit Max 32, siguiendo con la filosofía de software libre y dispositivos de bajo coste. Estas placas están programadas con un firmware único para todas, capaz de hacer trabajar a cada placa como nodo esclavo o maestro, aportando al sistema total flexibilidad para cambiar la placa o cambiar su configuración sin alterar el funcionamiento de la red.

Con todo esto, se ha conseguido crear un prototipo con visión hacia a un sistema que favorezca el control de la calidad del proceso de fermentación del vino mediante la captura de datos para su posterior análisis. A través de este análisis, se podrán crear estrategias o realizar modificaciones en futuras elaboraciones aportando de esta manera cierto valor añadido al producto final.

Líneas de continuación

Mediante el desarrollo de este prototipo se ha creado una base para el desarrollo de un sistema que pueda implementarse físicamente en un proceso real. Para ello se tendrían que desarrollar aspectos como:

- Estudio y elección de sensores adecuados para un proceso en concreto.
- Desarrollo de una shield que permita la conexión directa de la placa con el bus CAN.
- Diseño de una caja industrial para proteger al nodo y la shield, con las características necesarias para poder funcionar en entornos industriales.
- Mejoras al programa en VB.NET, como:
  - La posibilidad de configurar un mayor número de nodos.
  - La capacidad de poder guardar y cargar distintas configuraciones del sistema.

Otra mejora interesante de la aplicación de usuario es la implementación de un sistema de alertas. A través de él, el usuario podría establecer unos valores de referencia máximos y mínimos para cada variable del proceso. Si el valor medido quedase fuera de esos valores de referencia durante un tiempo establecido por el usuario, se trasmitiría un mensaje SMS o un correo electrónico de alerta a través de un módem vía GSM.

Para el desarrollo de esta funcionalidad, es necesario crear un nuevo formulario en el cual el usuario introduzca los valores entre los que debe estar comprendida cada variable del proceso. Además, debe permitir establecer el tiempo que tiene que permanecer el valor fuera de los establecidos para que se genere la alerta.

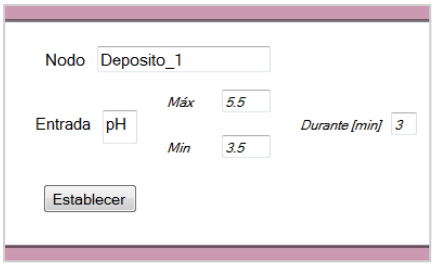


Ilustración 53- Formulario de la aplicacion de usuario

Actualmente la aplicación permite realizar 25 mediciones al día para cada entrada. Por ello sería necesario realizar una monitorización continua del valor de las entradas, pero registrando únicamente en la base de datos los valores medidos a las horas establecidas por el usuario.

Si se mantuviera un valor fuera de los valores de referencia durante un tiempo mayor al establecido, la aplicación mandaría vía serie al nodo maestro el valor, la entrada a la que corresponde y el nodo al que pertenece la entrada. A través de un módem USB conectado al nodo maestro, éste sería capaz de transmitir al usuario un SMS o un correo electrónico con los detalles de la alerta.

Ilustración 54- ChipKIT Network Shield con módem USB

La dificultad de esta funcionalidad cae en la programación de la aplicación, ya que para la parte de la transmisión de la alerta existen librerías y shields USB para ChipKit que simplifican el desarrollo de esa parte.

2.10 PLANIFICACIÓN

En este apartado se va a indicar la lista de tareas que se han llevado a cabo para la consecución del prototipo y su desarrollo en el tiempo.

TAREA 1	**Definición del proyecto y planteamiento de objetivos**	**7**
TAREA 2	**Instalación de software**	**11**
	MySQK Workbench	3
	Visual Studio 10, con extensiones Excel y MySQL	4
	MPIDE,con librerías correspondientes	4
TAREA 3	**Aprendizaje autónomo**	**76**
	Consultas a bases de datos	8
	Pogramación en VB.NET	25
	Programación en Arduino	15
	Funcionamiento de redes CanBUS	8
TAREA 4	**Comunicación serie chipKIT - Visual Basic**	**5**
TAREA 5	**Comunicación Visual Basic - MySQL**	**7**
TAREA 6	**Comunicación entre nodos CAN bus**	**8**
TAREA 7	**Protocolo para la comunicación entre nodo maestro - nodo esclavo**	**20**
TAREA 8	**Desarrollo del firmware de las placas**	**35**
TAREA 9	**Desarrollo de la aplicación Visual Basic**	**70**
TAREA 10	**Puesta en marcha del prototipo**	**15**
TAREA 11	**Documentación del proyecto**	**90**
TAREA 12	**Presentación del proyecto**	**25**
	TOTAL [horas]	**364**

Tabla 17 - Planificación del proyecto

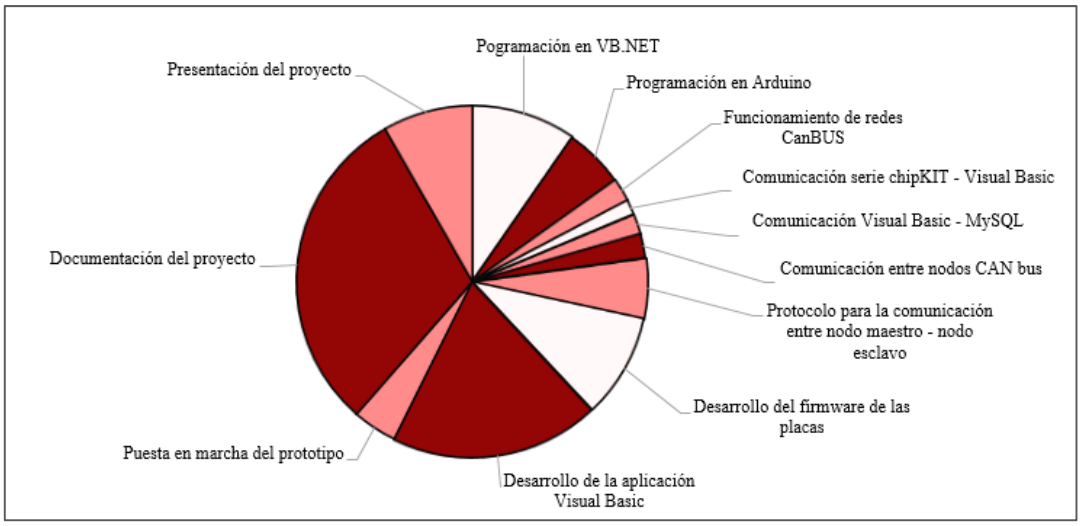


Ilustración 55 – Proporción de duración de cada actividad sobre la duración total

2.11 ORDEN DE PRIORIDAD ENTRE LOS DOCUMENTOS BÁSICOS

El orden de prioridad de los documentos en este proyecto se ha establecido conforme a la norma UNE 17001:2014.

1. Planos
2. Pliego de condiciones
3. Presupuesto
4. Memoria

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

3. ANEXOS

Daniel Pérez García

Septiembre 2018

INDICE DE LOS ANEXOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

3.1 PLACA CHIPKIT MAX32

3.1.1 Descripción general

La placa ChipKIT Max32 está basada en el microcontrolador PIC32MX795F512L, perteneciente a la familia de microcontroladores PIC32 de 32 bits de Microchip. La Max32 está basada en la placa Arduino Mega 2560, lo que la hace compatible con muchos shields de Arduino. Cuenta con una interfaz USB para su conexión al IDE y se puede alimentar a través de esta conexión o de una fuente o batería externa. Es fácil de usar y es adecuada tanto para usuarios principiantes como para avanzados para su empleo en sistemas de control electrónicos y embebidos.

Su entorno de desarrollo integrado multiplataforma (MPIDE) está basado den el IDE Arduino original con modificaciones para poder ser compatible con un PIC32. Este microcontrolador proporciona a la placa una MAC Ethernet 10/100, un controlador OTG de velocidad USB 2.0 y dos controladores CAN. Pasa usar estos periféricos es necesario una placa complementaria con la chipKIT Network Shield, o bien hardware adicional.

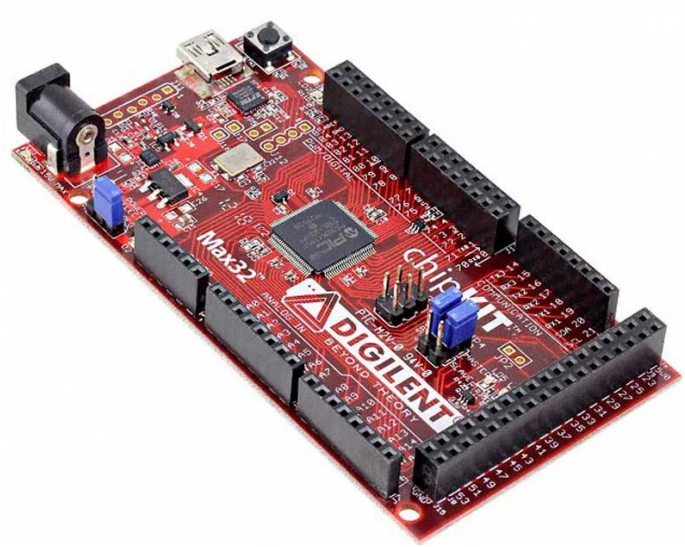


Ilustración 1 - Placa chipKIT Max32

Algunas características de esta placa son:

- Microcontrolador Microchip PIC32MX795F512L (80 Mhz, Flash 512K, 128K RAM)
- Tensión de funcionamiento de 3.3V
- Corriente de funcionamiento típica de 90mA
- Tensión de entrada recomendada de 7 a 15 V
- Tensión de entrada máxima de 20V

- 83 pines disponibles de E/S
- 16 entradas analógicas
- Mac Ethernet 10/100
- Controlador USB 2.0
- 2 controladores CAN

3.1.2 Elementos de la placa

La placa presenta las siguientes características de hardware:

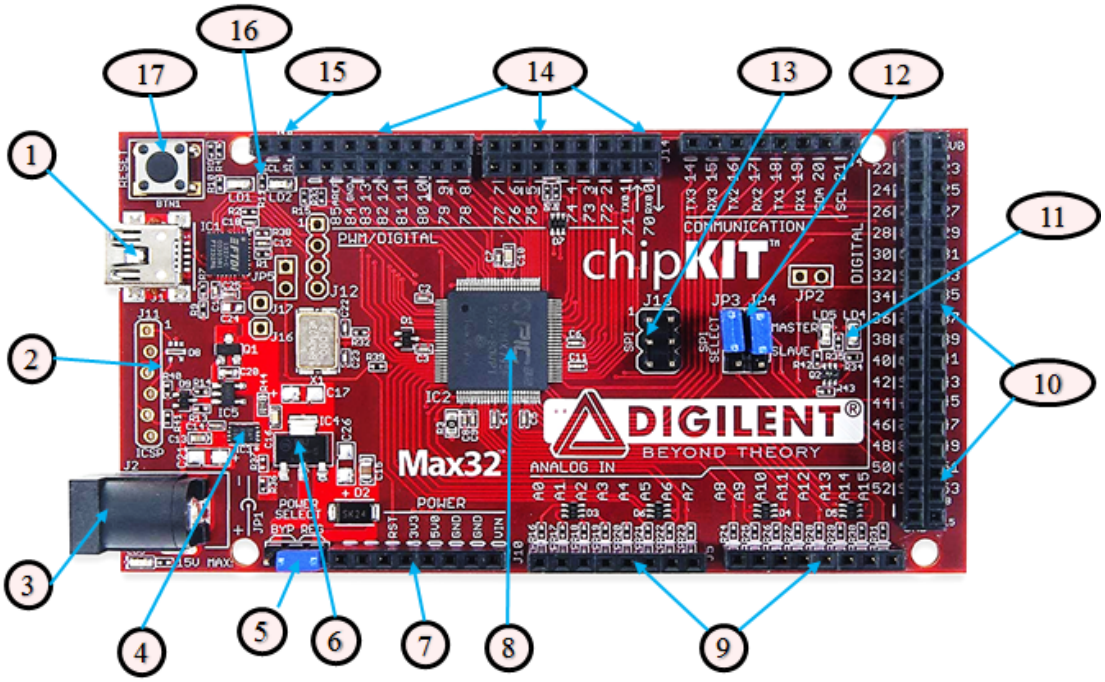


Ilustración 2 - Componentes de la placa chipKIT Max32

1) **Conector USB**: se conecta a un puerto USB en el PC para establecer comunicación vía serie con el entorno MPIDE. También sirve como alimentación de la placa

2) **Conector de herramienta de depuración**: se usa para conectar las herramientas de programación/depuración de Microchip. Esto permite a la placa trabajar coo una placa de desarrollo tradicional a través del IDE de Microchip.

3) **Conector de alimentación externa**: La placa se alimenta externamente a través de un conector Barrel Jack de 5.5mm x 2.1mm.

4) **Fuente de alimentación - Regulador de tensión a 3.3V**: puede proporcionar hasta 500mA.

5) **Jumper de selección de alimentación:** dirige la alimentación desde el conector de alimentación a través del regulador de tensión de 5V. En posición de bypass se omite este regulador, siendo la tensión máxima aplicable a la placa de 6V.

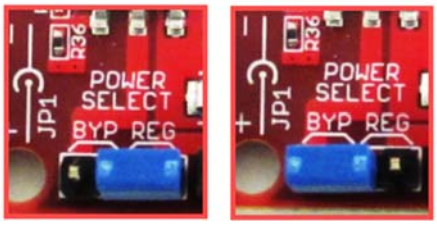


Ilustración 3 - Jumpers de alimentación de la placa

6) **Fuente de alimentación - Regulador de tensión de 5V**: Se usa para alimentar el regulador de 3.3V y las shields de expansión. Puede proporcionar hasta 800 mA.

7) **J10 – Alimentación de shields de expansión**: este conector proporciona energía a las placas de expansión.

8) **Microcontrolador PI32**: procesador principal de la placa

9) **J5, J7 – Conectores de señal analógica**: dan acceso a los pines de E/S analógicas/digitales.

10) **J6, J8, J9, J15 - Conectores de señal digital y alimentación:** proporcionan 5V de potencia, tierra y acceso a los pines de E/S digitales.

11) **Led de usuario**: este led esta conectado a la salida digital del pin 13.

12) **JP3, JP4 – Jumpers SPI Maestro/Esclavo**: cambian las señales SPI para el uso de la placa como dispositivo maestro SPI o esclavo SPI.

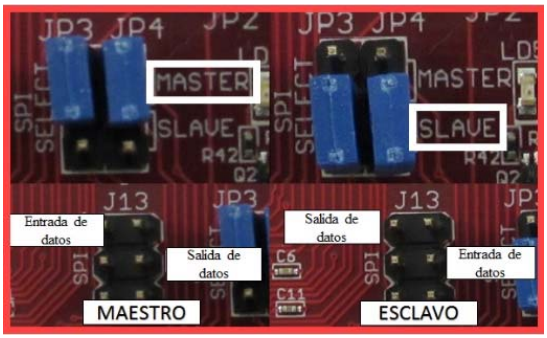


Ilustración 4 - Jumpers de configuración SPI

13) **J13 – Conector de señal SPI:** proporciona acceso alternativo a las señales SPI.

14) **J3, J4, J14 – Conectores de señal digital**: proporcionan acceso a los pines de E/S digitales.

15) **J18 - I^2C:** señales dedicadas I^2C. Se conectan al I2C1 del microcontrolador y se comparten con los pines 20 y 21.

16) **Leds de estado de comunicaciones**: indican actividad en la interfaz serie USB.

17) **Botón reestablecer**: reinicia el microcontrolador.

3.1.3 Descripción del hardware

MPIDE y comunicación serie USB

Esta placa está diseñada para su uso con el IDE multiplataforma MPIDE. Digilent produjo esta plataforma a partir del IDE de Arduino. Por ello es completamente compatible con versiones anteriores.

El MPIDE utiliza un puerto de comunicaciones via serie para comunicarse con un gestor de arranque (bootloader) que se ejecuta en la placa. El puerto serie de la placa está implementando usando el convertidos serie USB FTDI FT232R. Antes de utilizar el el MPIDE para comunicarse con la placa, se debe instalar en el PC el driver de la placa apropiado. Este proceso se suele realizar automáticamente al conectar la placa al PC. La conexión de la placa es un conector mini-USB estándar.

Ilustración 5 - Entorno MPIDE de chipKIT

Cuando el MPIDE necesita comunicarse con la placa, ésta se reinicia y comienza a ejecutar el bootloader. Cuando el MPIDE abre la conexión serie del PC, el pin DTR del chip FT232R se desactiva. Este pin se conecta a través de un condensador al pin MCLR del microprocesador. Con un nivel bajo en el pin MCLR el microprocesador se resetea, reiniciando el bootloader. Este reinicio automático se puede deshabilitar a través de una pista que se puede desconectar en la parte inferior de la placa entre los pines JP5. Si esta pista se desconecta, el reset automático se puede reestablecer cargando JP5.

Cuando se envían o reciben datos entre el PC y la placa, dos leds rojos (LD1 y LD2) parpadean.

Fuente de alimentación

Esta placa está diseñada para ser alimentada a través de USB o de una fuente externa. Hay un circuito de conmutación automática que en caso de que se usen ambos suministros use el externo.

La parte de alimentación usa dos retuladores de tensión. El primero regula la tensión externa a 5V para alimentar el bus VCC5V0. El segundo regula la tensión de este bus a 3.3V para alimentar al bus VCC3V3, que alimenta el PIC32.

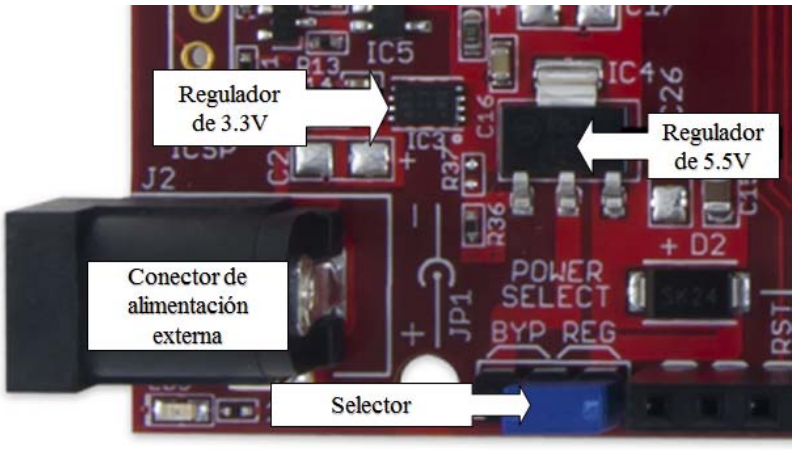


Ilustración 6 - Detalle del circuito de alimentación

Para proteger el circuito de alimentación externa hay un diodo de polaridad inversa. Teniendo en cuenta la caída de tensión en el diodo más la caída en el regulador, la tensión mínima de entrada al regulador debe de ser de 7V para generar una salida fiable de 5V. La tensión máxima admitida es de 20V, pero el máximo recomendado es 15V. El regulador protección térmica y contra cortocircuitos, y se apaga automáticamente para evitar daños.

En el caso del regulador de 3.3V, soporta una corriente de salida máxima de 500 mA. La tensión máxima de entrada para este regulador es de 6V. Al igual que el anterior, tiene protección térmica y contra cortocircuitos.

El bus VCC5V0 de 5V puede alimentarse desde tres fuentes:

- El bus USB5V0 cuando la placa está funcionando con alimentación USB
- La salida del regulador de 5V
- Directamente desde el suministro externo cuando se opera en posición BYP

Como se ha comentado antes, el cambio de alimentación USB alimentación externa se realiza automáticamente, empleando la externa si ambas están presentes.

Cuando el jumper del selector esté en posición de bypass (BYP), se podrá alimentar externamente la placa desde una fuente de entrada de al menos 3.5V. Esto permitirá alimentarse a la placa con baterías o fuentes de alimentación de menor voltaje. En este caso no se debe aplicar más de 6V de alimentación, ya que se podría destruir el regulador de 3.3V y en consecuencia el microprocesador PIC32.

El microprocesador admite un máximo de 98mA cuando opera a 80Mhz. Esto permite hasta aproximadamente 400mA desde el bus VCC3V3 y hasta 700mA desde el bus VCC5V0 para alimentar dispositivos externos.

A través del contector J10 se proporciona alimentación a las shields conectadas a la placa. Presenta lo siguientes pines:

- **NC**: no se usa.
- **IOREF**: proporciona una tensión de referencia de 3.3V.
- **P32_MCLR**: conecta el pin MCLR al PIC32 pudiendo reiniciarlo desde este pin.
- **VCC3V3**: conecta el bus de 3.3V a las shields. Proporciona hasta 400mA.
- **VCC5V5**: conecta el bus de 5V a las shields. Proporciona hasta 700mA.
- **GND**: conexión a tierra común entre la placa y los shields.
- **VIN**: proporciona alimentación no regulada a la shield.

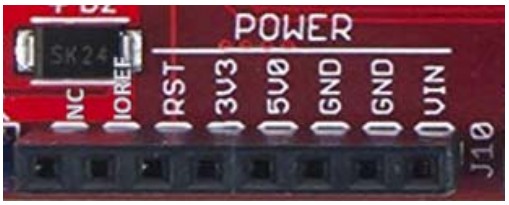


Ilustración 7- Detalle del conector J10

Compatibilidad 3.3V-5V

El PIC32 opera a 3.3V, y las placas de Arduino originales funcionan 5V al igual que muchas shields. Cuando se trata de compatibilidad de 5V para una lógica de 3.3V hay que considerar la protección de las entradas de 3.3V contra el daño causado por señales de 5V. Otra consideración es si la salida de 3.3V es suficientemente alta para ser reconocida como un valor lógico alto por una entrada de 5V, ya que algunos dispositivos lógicos la reconocen y otros no.

Los pines de E/S digitales en el PIC32 admiten 5V, pero los pones analógicos no. Para proporcionar tolerancia a esos pines, la placa presenta diodos de protección y resistencias limitadoras de corriente para protegerlos de voltajes de entrada de 5V.

Por ello es seguro aplicar valores lógicos de 5V a la placa sin riego de dañar el microprocesador.

Pines de Entrada/Salida

La Max32 proporciona 83 pines de E/S desde el microprocesador PIC32 a los conectores J3, J4, J5, J7, J8, J9 y J14.

El microprocesador puede aportar o descargar un máximo de 18 mA en todos los pines de E/S digitales. Sin embargo, para mantener la tensión de salida en el rango especificado, la corriente de cada pin se debe limitar a +7/-12 mA. La corriente máxima admitida a través de todos los pines de manera simultanea es de +/-200 mA, y la tensión máxima que se puede aplicar a cada pin es de 5.5V.

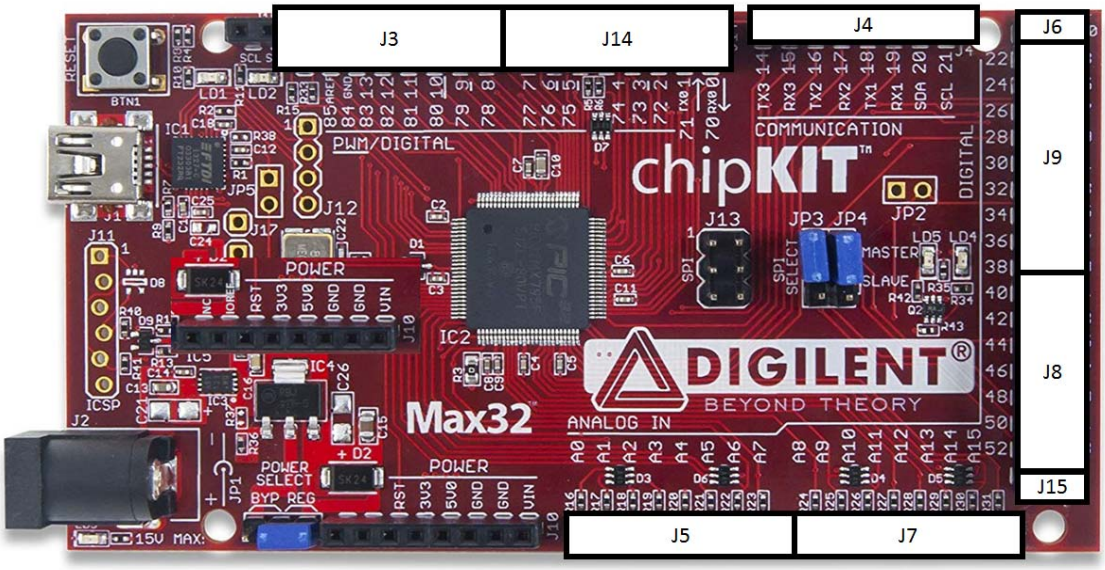


Ilustración 8- Conectores de E/S de la placa

Las características de los conectores son las siguientes:

- J3, J8, J9, J14: son conectores hembra de 2x8 pines. Corresponden a E/S digitales.
- J5, J7: son conectores hembra de 1x8 pines. Corresponden a E/S digitales y analógicas.
- J4: es un conector de pin hembra de 1x8 pines. Corresponde a E/S digitales y a señales de comunicación UART1-3 e I^2C.
- J6, J15: son conectores de dos pines que proporcionan alimentación y tierra a las shields. J6 contiene dos pines conectados al bus VCC5V0, y J15 dos pines conectados a GND.

Periféricos de Entrada/Salida

El microprocesador proporciona una serie de funciones de periféricos:

- PWM: salida modulada por ancho de pulso. Pines 3, 5, 6, 9 y 10.
- Led de usuario: pin 13 y pin 86.
- Puerto 0 de la UART: puerto serie asíncrono. Corresponde al pin 0 (RX0), pin 1 (TX0). Estos pines están conectados al conector J14 y al convertidor serie USB

FT232R. Se pueden usar para conectar un dispositivo serie externo si no se está usando la interfaz de comunicación USB.

- Puerto 1 de UART: puerto serie asíncrono. Corresponde al pin 19 (RX1), pin 18 (TX1).

- Puerto 2 de UART: puerto serie asíncrono. Corresponde al pin 17 (RX2), pin 16 (TX2).

- Puerto 3 de UART: puerto serie asíncrono. Corresponde al pin 15 (RX3), pin 14 (TX3).

- SPI: puerto serie síncrono. Corresponde al pin 53 (SS), pin 51 (MOSI), pin 50 (MISO), pin 52 (SCK). Estas señales también aparecen en el conector J13. Los jumpers JP3 y JP4 se emplean para seleccionar la placa como maestro (MOSI trasnmite – MISO recibe) o como esclavo (MISO transmite – MOSI recibe).

- I^2C: interfaz serie síncrona. Estas señales están disponibles en el conector J18 y se comparten con los pones 21(SCL) y el pin 20 (SDA) en el conector J4.

- Interrupociones externas: pin 3 (INT1), 7 (INT2), 21 (INT3) y 20 (INT4).

- Referencia del convertidor A/D: pin externo más a la izquierda del conector J3. Proporciona una referencia de tensión para determinar el rango de voltaje de entrada de los pines analógicos.

- RESET: botón que reinicia el microcontrolador.

- RTTC: reloj/calendario en tiempo real. La entrada de reloj corresponde al pin 75.

.

3.2 CANBUS

3.2.1 Historia

El protocolo CAN (Controller Area Network) tiene su origen en 1986, cuando Bosch presentó el bus CAN en la Sociedad de Ingenieros de Automoción (SAE) para la transmisión de mensajes entre unidades de control electrónicas dentro del automóvil, convirtiéndose en un protocolo de comunicaciones de éxito. Este sistema permitió una reducción considerable de la complejidad del cableado ya que previo a esto, cada dispositivo electrónico iba cableado mediante punto a punto. El reemplazo del cableado por una red de comunicación provocó una reducción importante en peso y coste, tanto de cableado como de los sensores utilizados.

A pesar de estar inicialmente enfocado al entorno de la automoción, hoy en día también se utiliza en otros tipos de vehículos como trenes o barcos, así como distintos procesos industriales.

3.2.2 Características

Es un protocolo de comunicación en serie basado en la topología bus para el intercambio de información en tiempo real en sistemas distribuidos. La característica principal de este sistema es que es un protocolo orientado a mensajes, es decir, el mensaje no va dirigido a ningún nodo o unidad de mando en concreto. Dentro de la red, cada mensaje tiene un identificador el cual permite a cada unidad reconocer si ese mensaje le interesa o no y en consecuencia aceptarlo.

Toda unidad de mando puede ser tanto emisora como receptora. Al ser un sistema de funcionamiento en tiempo real cualquier unidad puede transmitir a través del bus siempre y cuando esté libre. Para solucionar el conflicto de que más de una unidad intente transmitir al mismo tiempo se establece una prioridad, la cual viene definida en el identificador del mensaje. De esta manera, mediante este mecanismo de arbitraje el sistema asegura que mensajes más prioritarios se transmitan antes en función de su identificador.

Se trata de un sistema robusto en cuanto a la consistencia en la transmisión de datos. Esto se lleva a cabo gracias a una serie de mecanismos que permiten una detección y señalización de errores muy fiable. Cuando se produce un error en la transmisión de un mensaje, éste se anula y se vuelve a transmitir. Por otro lado, si se detecta un fallo en alguna unidad de mando que pueda comprometer la integridad de la red, el propio sistema deja esa unidad fuera de servicio, pero la red puede seguir funcionando.

3.2.3 Capas de la pila OSI

Muchos protocolos de red se describen utilizando el modelo de interconexión de sistemas abiertos (OSI) de siete capas. La especificación del protocolo CAN está publicada en el estándar ISO 11898. En esta norma se define la capa o nivel de enlace de datos y parte de la capa física en el modelo OSI. La propia estandarización diferencia estas dos capas. En este protocolo, ISO 11898-1 define la capa de enlace de datos mientras que en la capa física depende de la velocidad siendo ISO 11898-2 para alta velocidad e ISO 11898-3 para baja.

La capa física define el estado del medio físico, así como sus características tanto eléctricas como materiales para la transmisión de datos entre nodos a través del bus. Se puede dividir en tres subcapas.

- Physical Signaling Layer (PSL). Es la encargada de codificar/decodificar así como sincronizar y transmitir/recibir la información.
- Physical Medium Attachment (PMA). Acondiciona los niveles lógicos de transmisión y recepción a los establecidos en el protocolo. Amplifica la señal cuando se vuelca en el bus y la atenúa cuando se envía al controlador.
- Medium Dependent Interface (MDI). Define el tipo de conector y el medio de transmisión de la información.

El nivel de enlace de datos es el responsable de que la transferencia de información se realice de manera fiable, es decir, que la información fluya libre de errores entre unidades conectadas directamente. Esta tarea la realiza a través de la subcapa Medium Access Control (MAC) y Logical Link Control (LLC). MAC se encarga de la estructuración de la información en forma de tramas y la detección y control de errores. Por otro lado, LLC gestiona el control y notificación de sobrecarga de datos, filtrado de mensajes y la recuperación del sistema ante fallos.

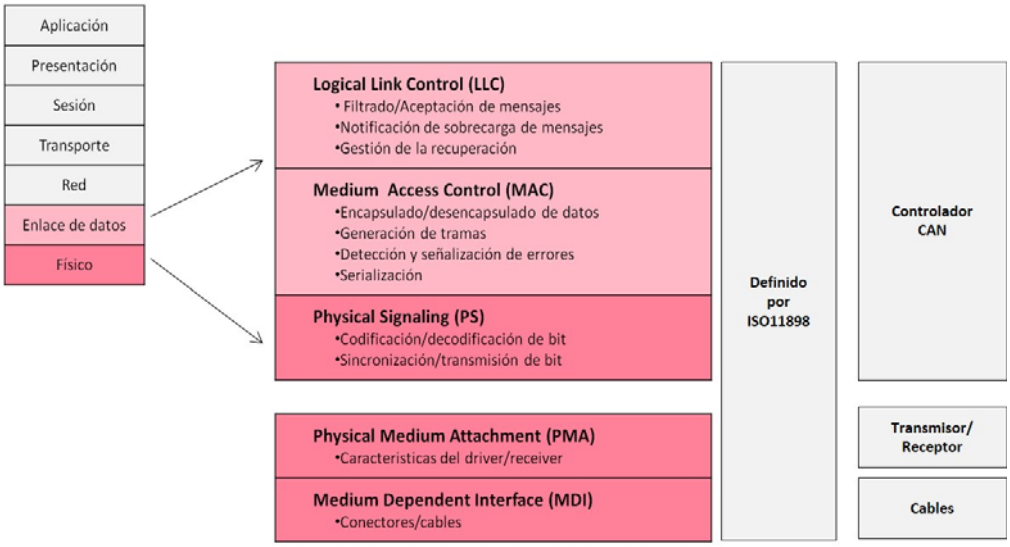


Ilustración 9 - Capas de la pila OSI en el protocolo CAN

3.2.4 Tipos de bus CAN

Como se ha comentado en el apartado anterior, la especificación CAN viene recogida en el conjunto de estándares ISO 11898. En base a estos estándares, se puede clasificar en CAN de alta velocidad y de baja velocidad tolerante a fallos.

CAN de alta velocidad

Permite configurar velocidades de 40 Kbit/s hasta 1 Mbit/s en función de la longitud del cable. Emplea un único bus lineal con una resistencia de 120 Ω en cada extremo, lo que permite una conexión simple entre las unidades. Esta configuración de la capa física viene recogida en el estándar ISO 11898-2, pero existen algunas extensiones compatibles que resultan útiles para sistemas con requisitos específicos. Estas extensiones están definidas en sus respectivos estándares y son interoperables entre sí, es decir, pueden coexistir en la misma red.

- ISO 11898-5. Define la capa física de alta velocidad para sistemas que necesiten un consumo de energía bajo cuando no hay comunicaciones en el bus.
- ISO 11898-6. Esta extensión proporciona un método selectivo para la activación de los nodos mediante tramas configurables.

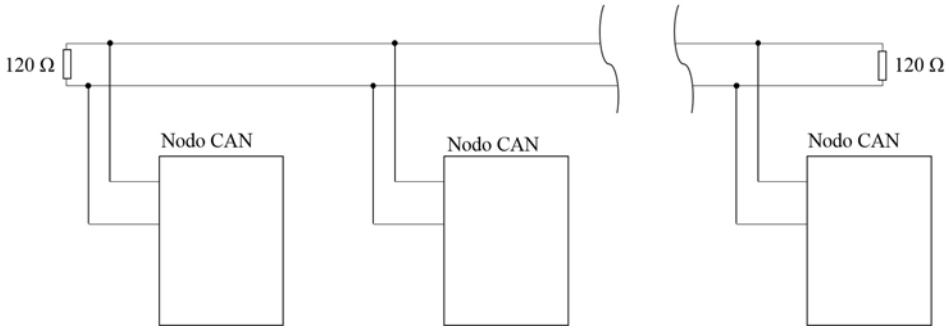


Ilustración 10 - Configuración de alta velocidad a través del bus lineal

CAN de baja velocidad tolerante a fallos

Esta configuración permite utilizar un bus lineal, un bus en estrella o varios buses en estrella conectados por un bus lineal. Ofrece velocidades de transmisión que van desde 40 Kbit/s hasta 125 Kbit/s. Esta especificación permite mantener la comunicación con bus CAN en caso de que se presente una falla en el cableado en las líneas. En este tipo de red, cada dispositivo tiene su propia resistencia de terminación, cuyo valor total debe rondar los 100 Ω sin ser inferior a ese valor.

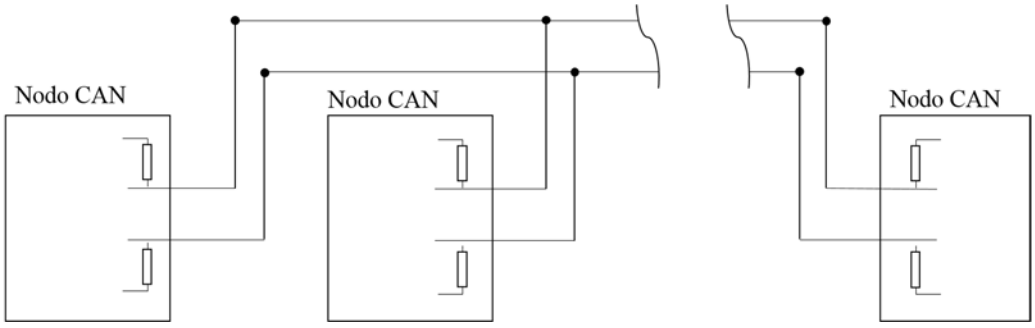


Ilustración 11- Configuración de baja velocidad en un bus estrella

3.2.5 Bus

Los datos fluyen de manera bidireccional por el bus formado por dos cables trenzados apantallados o sin apantallar. Al ser trenzado ofrece una protección ante posibles interferencias externas producidas por posibles campos magnéticos. Se puede conseguir una mayor protección si además está apantallado, pero esto conlleva un aumento del coste del cableado.

La trasmisión de mensajes se realiza de modo diferencial entre estos dos cables, cuyas señales se denominan CAN-H y CAN-L (señal de nivel lógico alto y bajo respectivamente). Para alta velocidad los valores de tensión de cada señal oscilan entre 1,5-2,5V para CAN-L y 2,5-3,5V para CAN-H , mientras que para baja velocidad oscilan entre 0-2,75V y 2,25-5V para CAN-H y CAN-L respectivamente.

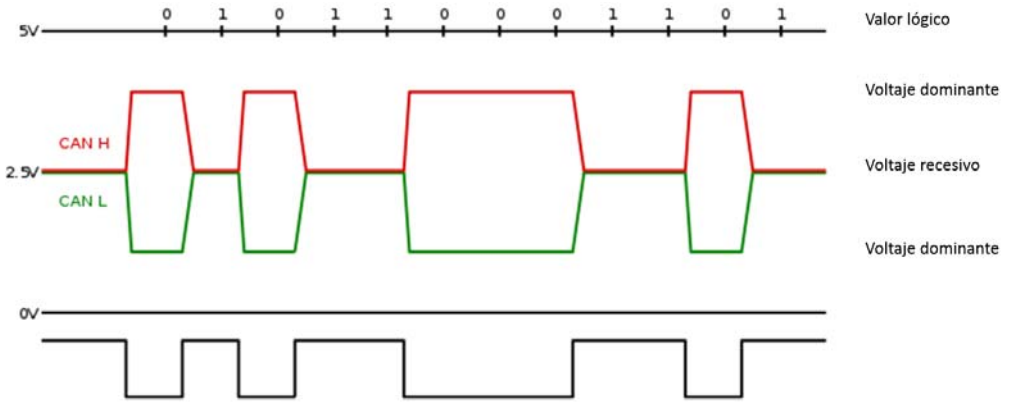


Ilustración 12 - Formato de señales en configuración de alta velocidad

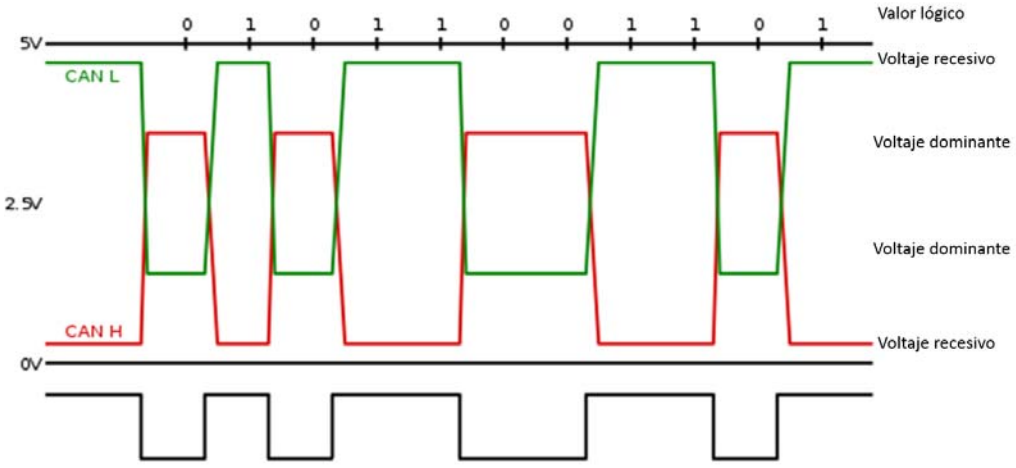


Ilustración 13- Formato de señales en configuración de baja velocidad

En cuanto al conector, este estándar no especifica ningún tipo en concreto por lo que se puede escoger el más adecuado para cada aplicación.

3.2.6 Resistencias de cierre

La impedancia característica de la red ronda los 120 Ω. Para ello se colocan en ambos extremos de los cables resistencias cuyo valor ronde esa impedancia (ver 3.1.4 Tipos de bus CAN). Su función es la de adecuar el funcionamiento de la red para diferentes longitudes, disposiciones y número de unidades, ya que impiden posibles efectos parásitos que puedan alterar el mensaje.

3.2.7 Controlador

Es el elemento que se encarga de la comunicación del microprocesador del módulo con el transceiver acondicionando la información que entra y sale entre ambos componentes.

El controlador interviene en la sincronización entre los distintos módulos. Para ello almacena los bits recibidos desde el bus hasta que esté disponible el mensaje completo, que luego puede ser recuperado por el microprocesador. Por otro lado, cuando se trata de enviar un mensaje, es el encargado de transmitir los bits al bus cuando el bus está libre. Trabaja con muy bajos niveles de tensión.

También es el encargado de determinar la velocidad de transmisión de los mensajes, la cual dependerá de cada aplicación. Por ejemplo, en un automóvil en la línea CAN de control para el ABS la velocidad ronda los 500 Kbit/s, mientras que en la línea que gobierna el sistema de confort se transmite a unos 62,5 Kbit/s.

Cada módulo del sistema va dotado de su propio controlador.

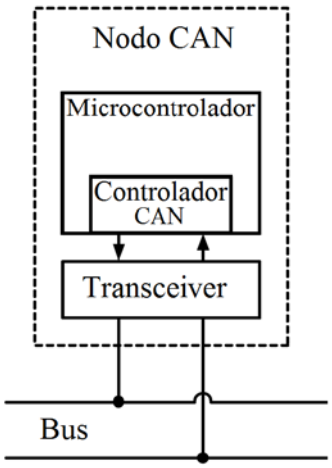


Ilustración 14 - Elementos de un nodo CAN

3.2.8 Transceiver

Cuenta con un transmisor y un receptor. Es el elemento a través el cual se envían y reciben los mensajes. Su misión es convertir la secuencia de datos procedentes del controlador CAN a los niveles del bus, así como acondicionar y preparar la información procedente del bus para que pueda ser utilizada por los controladores. Para ello amplifica la señal cuando la información se vuelca en la línea y la reduce para suministrarla al controlador.

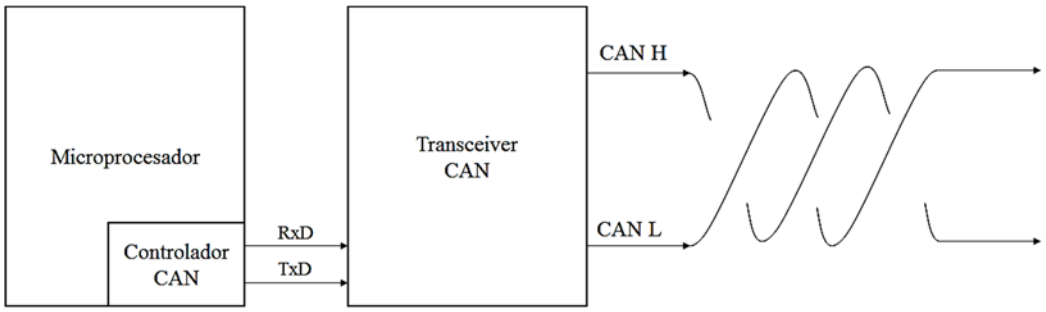


Ilustración 15 - Controlador – Transceiver

3.2.9 Trama CAN

Los mensajes se estructuran en tramas para su transmisión. Las tramas son básicamente sucesiones de 0 y 1 los cuales representan los niveles de tensión CAN H y CAN L en los cables del CAN bus.

Las tramas poseen diferentes campos que informan de diferentes aspectos del mensaje. A través de estos campos se identifica la unidad de control, el principio y final del mensaje o el tamaño de los datos transmitidos entre otras cosas. Además, las tramas utilizan bit stuffling, es decir, cuando aparecen 5 bits consecutivos del mismo valor, se introduce un bit extra de valor contrario para evitar la desincronización.

Como se verá en el próximo apartado, existen dos tipos fundamentales de trama en base al número de bits del identificador: estándar y extendida.

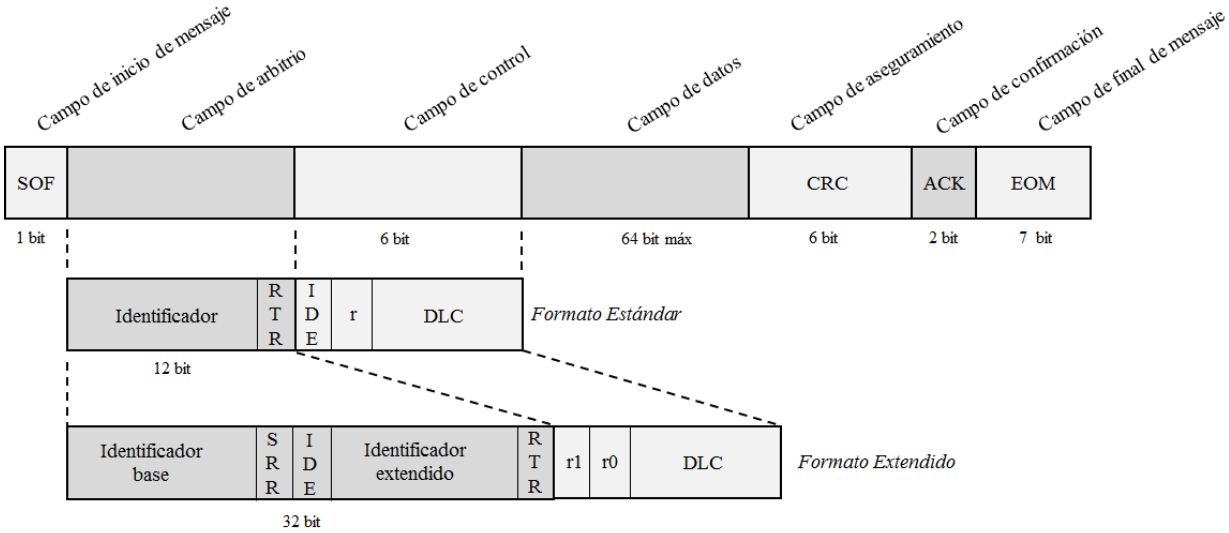


Ilustración 16 - Estructura de la trama CAN

Campos de la trama CAN bus

Campo de inicio de mensaje

También denominado SOF (Start Of Frame). Está formado por un bit de estado dominante. Las unidades de mando se sincronizan entre sí a través de un flanco descendente de este bit.

Campo de arbitrio

Está formado por 12 o 32 bits depende de si la trama es estándar o extendida. Dentro de este campo se encuentre el identificador, a través del cual se indica la prioridad del nodo dentro de la red. Un nodo con mayor prioridad será aquel cuyo identificador tenga el valor más bajo.

En este campo también se encuentra el bit RTR, que indica si la trama es remota (RTR=1) o es una trama de datos (RTR=0). En el siguiente apartado se explicará con mas detalle los tipos de tramas.

Campo de control

Este campo está formado por 6 bits. El bit IDE indica si se trata de una trama remota (IDE=0) o de una trama extendida (IDE=1). El bit RB0 es un bit recesivo (1) reservado por el protocolo. Los otros 4 bits que conforman el DLC indican el número de bytes que forman el campo de datos.

Campo de datos

En este campo están contenidos los datos del mensaje. Puede estar formado por un máximo de 8 bytes, según se indique en el DLC del campo de control.

Campo de aseguramiento

Este campo es el encargado de la detección de errores en la transmisión del mensaje. El último bit siempre es un bit recesivo (1) que delimita este campo CRC.

Campo de confirmación

Está formado por 2 bits. A la hora de transmitir un mensaje, en la trama del nodo transmisor el primer bit del campo ACK es recesivo (1). Cuando los nodos receptores reciben el menaje correctamente, envían un mensaje modificando este bit a estado dominante (0). De esta forma el nodo transmisor puede reconocer que alguna unidad de mando ha recibido el mensaje correctamente. En caso contrario, el nodo transmisor interpreta que el mensaje presenta algún error.

Campo de fin de trama

Está formado por 7 bits recesivos que indican el final del mensaje.

Tipos de trama CAN bus

Las tramas CAN se pueden clasificar según su función como trama de datos, trama remota y rama de saturación; y según el tamaño de su identificador como trama estándar y extendida.

Trama estándar

Tiene un identificador de 11 bits, en el cual se establece la prioridad de cada nodo en la red. Tras el identificador se encuentra el bit RTR mediante el cual se indica si la trama es remota o de datos.

Trama extendida

En esta trama, el identificador tiene 29 bits en total. En este caso tras la primera parte del identificador aparece el bit IDE que indica si la trama es estándar o extendida. Tras este bit aparece la segunda parte del identificador y el bit RTR.

Trama de datos

Esta trama se utiliza para el envío de datos a través de la red. Los datos se introducen en el campo de datos y pueden tener una extensión de hasta 8 bytes.

Trama remota

Los nodos tienen la capacidad de solicitar datos a otro nodo a través de este tipo de trama. El identificador de la trama debe ser el mismo que el nodo al cual se solicita la información. Además, el campo de datos estará vacío. Cuando nodo reciba la trama, éste enviará los datos.

Trama de error

Esta trama se genera al detectar un errar en la red por parte de un nodo. Esta trama está formada por un campo indicador de error y un delimitador de trama.

Trama de saturación

Esta trama se genera al detectar un bit dominante en el espacio entre tramas, o al no ser posible el envío de un mensaje debido a problemas internos de la red.

3.3 MCP 2551

3.3.1 Características del dispositivo

Es un transceptor CAN de alta velocidad que sirve de interface entre un controlador CAN y el bus físico. Proporciona la capacidad de transmisión y recepción diferencial al controlador CAN.

Ilustración 17 - Transceptor CAN MCP2551

Presenta las siguientes características:

- Soporta velocidades de operación de hasta 1MB/s
- Implementa los requisitos de la capa física del estándar ISO-11898
- Es compatible con sistemas de 12 y 24 V
- Se puede controlar la rampa externamente para emisiones RFI reducidas
- Detecta el fallo de tierra en la entrada TxD
- Reset activo y protección de caída de tensión
- Un nodo inoperativo no altera el bus CAN
- Baja corriente en modo de operación Standby
- Protección frente a daños producidos por cortocircuitos
- Protección frente a tensiones transitorias de alto voltaje
- Protección térmica con apagado automático

- Permite conectar hasta 112 nodos
- Fuerte inmunidad ante el ruido debido al bus diferencial
- Rango de temperatura industrial: -40ºC hasta +85ºC
- Rango de temperatura extendido: -40ºC hasta +125ºC

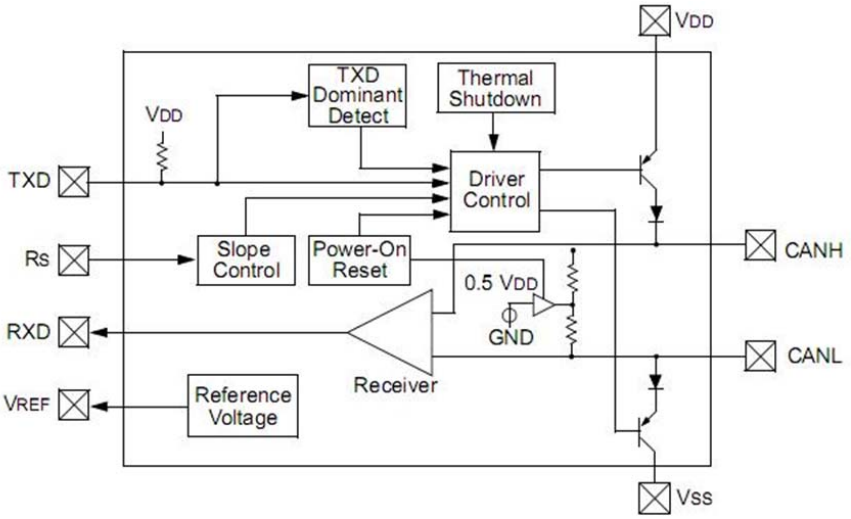


Ilustración 18- Diagrama de bloques del MDP2551

PIN	NOMBRE	FUNCIÓN
1	TXD	Entrada de transmisión de datos
2	VSS	Tierra
3	VDD	Tensión de alimentación
4	RXD	Salida de recepción de datos
5	VREF	Tensión de salida de referencia
6	CAN L	Tensión nivel bajo E/S CAN
7	CAN H	Tensión nivel alto E/S CAN
8	RS	Entrada de control de rampa

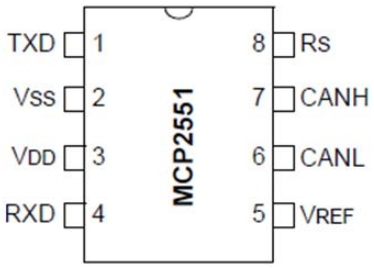


Ilustración 19- Pinout del MCP2551

Cada nodo en un sistema CAN debe tener un dispositivo como éste para convertir las señales digitales generadas por el controlador CAN a señales adaptadas para su transmisión por el bus (salida diferencial). Además, protege al controlador CAN de picos de alta tensión generados por fuentes externas.

Número de nodos de la red

El chip MCP2551 soporta una carga de 45Ω permitiendo hasta 112 nodos conectados (dada una mínima resistencia de entrada diferencial de 20k Ω y una resistencia de terminación de 120 Ω).

Función de transmisión

El bus CAN presenta dos estados: dominante y recesivo. El estado dominante aparece cuando la tensión diferencial entre CAN H y CAN L es mayor que una tensión de referencia definida. El estado recesivo, por el contrario, ocurre cuando esta tensión diferencial es menor que la definida. Estos estados corresponden con el nivel bajo y alto del pin TxD respectivamente.

Función de recepción

El pin de salida RxD refleja la tensión diferencial del bus entre la línea CAN H y CAN L. Los niveles bajo y alto de este pin corresponden al estado dominante y recesivo del bus, respectivamente.

Protección interna

Las líneas CAN H y CAN L están protegidas contra los cortocircuitos y transitorios eléctricos que puedan aparecer en el bus. Esta prestación evita la destrucción de la etapa de salida del transmisor durante un fallo.

Además, el dispositivo está protegido contra altas corrientes a través de un circuito térmico que deshabilita los drivers de salida cuando la temperatura supera los 165ºC. Todas las demás partes del chip se mantienen operativas, y la temperatura se disminuye disipando el calor por las salidas del transmisor. Esta protección es fundamental ante cortocircuitos en el bus.

Modos de operación

A través del pin Rs se pueden seleccionar tres modos de operación

- Alta velocidad
- Control de rampa
- Standby

Modo	Corriente en el pin Rs	Tensión resultante en el pin Rs
Standby	-10 IRS < 10 A	VRS > 0,7 VDD
Slope-Control	10 A < -IRS < 200 A	0,4 VDD < VRS < 0,6 VDD
Alta velocidad	-IRS < 610 A	0 < VRS < 0,3V DD

Tabla 1 - Modos de operación del chip MCP2551

En los modos de alta velocidad y control de rampa, los drivers para las señales CAN H y CAN L son regulados internamente para proporcionar una simetría controlada para minimizar la emisión de ruido.

Además, la rampa de la señal en CAN H y CAN L se puede controlar mediante una resistencia conectada en el pin R_S desde tierra. La pendiente de la rampa será proporcional a la corriente de salida en R_S. Esto podrá reducir aún mas las emisiones de ruido.

Alta velocidad

Este modo se selecciona conectando el pin R_S a V_{SS}. En este modo, los drivers de las salidas de transmisión tiempos de subida y bajada rápidos para poder soportar la alta velocidad de transmisión.

Control de rampa

Este modo reduce las emisiones de ruido limitando los tiempos de subida y bajada de CAN H y CAN L. La rampa, o velocidad de subida, se controla conectando una resistencia externa entre R_S y V_{OL} (normalmente tierra). Esta la pendiente de esta rampa será proporcional a la corriente de salida en el pin R_S. Ya que la corriente está determinada por esa resistencia, para cada valor de resistencia se alcanzará una velocidad de subida específica.

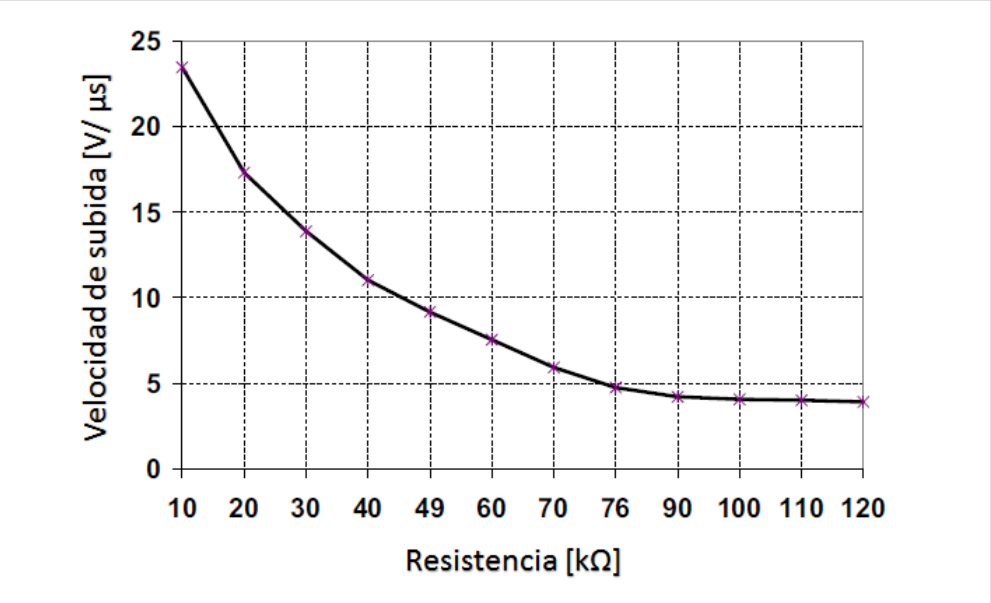


Ilustración 20 - Velocidad de subida vs Resistencia

Modo Standby

El dispositivo se puede llevar a modo Standby aplicando un nivel alto en el pin R_S. En este modo, el transmisor permanece desconectado y el receptor opera a una corriente mas baja. El pin de recepción del controlador (RxD) sigue funcional, pero a un ritmo más lento.

Detección permanente de estado dominante en TxD

Si el MCP2551 detecta un nivel bajo en la entrada TxD, se deshabilitarán los drivers de las salidas CAN H y CAN L para prevenir la corrupción de los datos en el can bus. Los drivers estarán deshabilitados si el nivel bajo permanece más de 1,25 ms. Esto implica un tiempo de bit máximo de 62,5 µs (16 kb/s), permitiendo transmitir hasta 20 bits consecutivos durante un bit de error múltiple. Los drivers permanecerán deshabilitados tanto como se mantenga el nivel bajo en TxD. Un flanco ascendente en TxD reseteará el temporizador y habilitará los drivers de CAN H y CAN L de nuevo.

Reset activo

Cuando se conecta el dispositivo, CAN H y CAN L permanecen en alta impedancia hasta que V_{DD} alcanza el nivel de tensión V_{PORH}. Además, CAN H y CAN L permanecerán en alta impedancia si TxD presenta un nivel bajo cuando V_{DD} alcanza V_{POHR}. CAN H y CAN L estarán activos solo cuando en TxD cese el nivel bajo. Una vez conectados, CAN H y CAN L volverán al estado de alta impedancia si la tensión en V_{DD} cae por debajo de V_{PORL}, protegiendo de esta manera al dispositivo ante sobretensiones.

3.3.2 Especificaciones CC

DC Specifications			Electrical Characteristics: Industrial (I): T_{AMB} = -40°C to +85°C V_{DD} = 4.5V to 5.5V Extended (E): T_{AMB} = -40°C to +125°C V_{DD} = 4.5V to 5.5V			
Param No.	Sym	Characteristic	Min	Max	Units	Conditions
Supply						
D1	I_{DD}	Supply Current	—	75	mA	Dominant; V_{TXD} = 0.8V; V_{DD}
D2			—	10	mA	Recessive; V_{TXD} = +2V; R_S = 47 kW
D3			—	365	µA	-40°C ≤ T_{AMB} ≤ +85°C, Standby;
			—	465	µA	-40°C ≤ T_{AMB} ≤ +125°C, Standby;
D4	V_{PORH}	High-level of the Power-on Reset comparator	3.8	4.3	V	CANH, CANL outputs are active when V_{DD} > V_{PORH}
D5	V_{PORL}	Low-level of the Power-on Reset comparator	3.4	4.0	V	CANH, CANL outputs are not active when V_{DD} < V_{PORL}
D6	V_{PORD}	Hysteresis of Power-on Reset comparator	0.3	0.8	V	

D7	$V_{CANH(r)}$; $V_{CANL(r)}$	CANH, CANL Recessive bus voltage	2.0	3.0	V	V_{TXD} = V_{DD}; no load.
D8	$I_{O(CANH)(reces)}$ $I_{O(CANL)(reces)}$	Recessive output current	-2	+2	mA	-2V < $V_{(CAHL,CANH)}$ < +7V, 0V <V_{DD} < 5.5V
D9			-10	+10	mA	-5V < $V_{(CANL,CANH)}$ < +40V, 0V <V_{DD} < 5.5V
D10	$V_{O(CANH)}$	CANH Dominant output voltage	2.75	4.5	V	V_{TXD} = 0.8V
D11	$V_{O(CANL)}$	CANL Dominant output voltage	0.5	2.25	V	V_{TXD} = 0.8V
D12	$V_{DIFF(r)(o)}$	Recessive differential output voltage	-500	+50	mV	V_{TXD} = 2V; no load
D13	$V_{DIFF(d)(o)}$	Dominant differential output voltage	1.5	3.0	V	V_{TXD} = 0.8V; V_{DD} = 5V 40W < R_L < 60W **(Note 2)**
D14	$I_{O(SC)(CANH)}$	CANH short-circuit output current	—	-200	mA	V_{CANH} = -5V
D15			—	-100 (typical)	mA	V_{CANH} = -40V, +40V. **(Note 1)**
D16	$I_{O(SC)(CANL)I}$	CANL short-circuit output current	—	200	mA	V_{CANL} = -40V, +40V. **(Note 1)**
D17	$V_{DIFF(r)(i)}$	Recessive differential input voltage	-1.0	+0.5	V	-2V < $V_{(CANL, CANH)}$ < +7V
			-1.0	+0.4	V	-12V < $V_{(CANL, CANH)}$ < +12V
Bus Line (CANH; CANL) Receiver: [TXD = 2V; pins 6 and 7 externally driven]						
D18	$V_{DIFF(d)(i)}$	Dominant differential input voltage	0.9	5.0	V	-2V < $V_{(CANL, CANH)}$ < +7V
			1.0	5.0	V	-12V < $V_{(CANL, CANH)}$ < +12V
D19	$V_{DIFF(h)(i)}$	Differential input hysteresis	100	200	mV	
D20	R_{IN}	CANH, CANL Common-mode input resistance	5	50	kW	
D21	$R_{IN(d)}$	Deviation between CANH and CANL Common-mode input resistance	-3	+3	%	V_{CANH} = V_{CANL}
Bus Line (CANH; CANL) Receiver: [TXD = 2V; pins 6 and 7 externally driven]						
D22	R_{DIFF}	Differential input resistance	20	100	kW	
D24	I_{LI}	CANH, CANL input leakage current	—	150	µA	V_{DD} < V_{POR}; V_{CANH} = V_{CANL} = +5V
Transmitter Data Input (TXD)						
D25	V_{IH}	High-level input voltage	2.0	V_{DD}	V	Output Recessive
D26	V_{IL}	Low-level input voltage	V_{SS}	+0.8	V	Output Dominant
D27	I_{IH}	High-level input current	-1	+1	µA	V_{TXD} = V_{DD}
D28	I_{IL}	Low-level input current	-100	-400	µA	V_{TXD} = 0V

Receiver Data Output (RXD)						
D31	V_{OH}	High-level output voltage	0.7 V_{DD}	—	V	I_{OH} = 8 mA
D32	V_{OL}	Low-level output voltage	—	0.8	V	I_{OL} = 8 mA
Voltage Reference Output (V_{REF})						
D33	V_{REF}	Reference output voltage	0.45 V_{DD}	0.55 V_{DD}	V	-50 µA < I_{VREF} < 50 µA
Standby/Slope-Control (R_S pin)						
D34	V_{STB}	Input voltage for standby mode	0.75 V_{DD}	—	V	
D35	I_{SLOPE}	Slope-control mode current	-10	-200	µA	
D36	V_{SLOPE}	Slope-control mode voltage	0.4 V_{DD}	0.6 V_{DD}	V	
Thermal Shutdown						
D37	$T_{J(sd)}$	Shutdown junction temperature	155	180	°C	
D38	$T_{J(h)}$	Shutdown temperature hysteresis	20	30	°C	-12V < V(CANL, CANH) < +12V

Tabla 2 - Especificaciones eléctricas CC del MCP2551

3.3.3 Especificaciones CA

AC Specifications			**Electrical Characteristics:** Industrial (I): T_{AMB} = -40°C to +85°C V_{DD} = 4.5V to 5.5V Extended (E): T_{AMB} = -40°C to +125°C V_{DD} = 4.5V to 5.5V			
Param No.	**Sym**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
1	t_{BIT}	Bit time	1	62.5	µs	V_{RS} = 0V
2	f_{BIT}	Bit frequency	16	1000	kHz	V_{RS} = 0V
3	TtxL2bus(d)	Delay TXD to bus active	—	70	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
4	TtxH2bus(r)	Delay TXD to bus inactive	—	125	ns	-40°C ≤ T_{AMB} ≤ +85°C, V_{RS} = 0V
			—	170	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
5	TtxL2rx(d)	Delay TXD to receive active	—	130	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
			—	250	ns	-40°C ≤ T_{AMB} ≤ +125°C, R_S = 47 kΩ
6	TtxH2rx(r)	Delay TXD to receiver inactive	—	175	ns	-40°C ≤ T_{AMB} ≤ +85°C, V_{RS} = 0V
			—	225	ns	-40°C ≤ T_{AMB} ≤ +85°C, R_S = 47 kΩ
			—	235	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
			—	400	ns	-40°C ≤ T_{AMB} ≤ +125°C, R_S = 47 kΩ

7	SR	CANH, CANL slew rate	5.5	8.5	V/µs	R_S = 47 kΩ,
10	t_{WAKE}	Wake-up time from standby (Rs pin)	—	5	µs	
11	TbusD2rx(s)	Bus Dominant to RXD Low (Standby mode)	—	550	ns	V_{RS} = +4V
12	$C_{IN(CANH)}$ $C_{IN(CANL)}$	CANH; CANL input capacitance	—	20 (typical)	pF	1 Mb/s data rate; V_{TXD} = V_{DD}
13	C_{DIFF}	Differential input capacitance	—	10 (typical)	pF	1 Mb/s data rate
14	TtxL2busZ	TX Permanent Dominant Timer Disable Time	1.25	4	ms	
15	TtxR2pdt(res)	TX Permanent Dominant Timer Reset Time	—	1	µs	Rising edge on TXD while device is in permanent Dominant state

Tabla 3 - Especificaciones eléctricas CA del MCP2551

3.3.4 Encapsulado

	Ref	Min [in]	Nom [in]	Max [in]
Número de pines	N		8	
Distancia entre pines	e		.100BSC	
Espesor total	A	-	-	0,210
Espesor 2	A2	0,115	0,130	0,195
Espesor 1	A1	0,015	-	-
Anchi	E	0,290	0,310	0,325
Ancho 1	E1	0,240	0,250	0,280
Distancia entre pines	D	0,348	365,000	0,400
Longitud de pines	L	0,115	0,130	0,150
Grosor de pines	c	0,008	0,010	0,015
Ancho parte gruesa del pi	b1	0,040	0,060	0,070
Ancho parte final del pin	b	0,014	0,018	0,022
Ancho total del chip	eB	-	-	0,430

Tabla 4 - Dimensiones del encapsulado

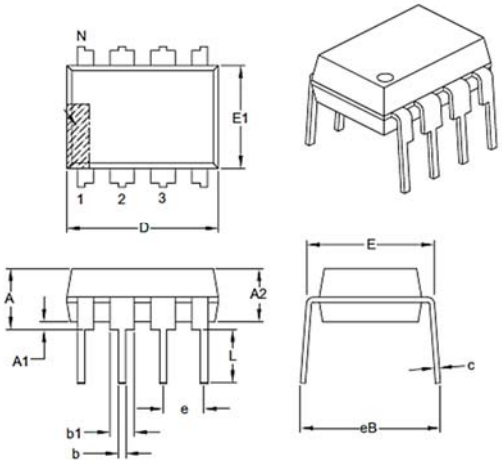


Ilustración 21 - Dimensiones del encapsulado

3.4 FUNCIONES VB.NET

En este apartado se van a presentar los principales eventos y funciones creadas en la aplicación Visual Basic.

Inicio y conexión con la base de datos

El primer formulario que aparece en la aplicación es el inicio de sesión del usuario de en la base de datos. Para la conexión con la base de datos hay que configurar parámetros como usuario, contraseña, servidor, puerto, etc. Esta información se toma a través de los *TextBox* del formulario.

El sistema puede funcionar sin estar conectado a la base de datos a través del Modo Buffer. Si realiza la conexión a la base de datos mientras el sistema está funcionando en Modo Buffer, al realizar la conexión se habilita el *Timer* que introduce los datos almacenados buffer Excel a la base.

```
    Private Sub bAcceso_Click(sender As Object, e As EventArgs) Handles
bAcceso.Click
        Try
            'Se configuran los parámetros de la conexión a la base de datos
            Variables.conexion.ConnectionString = "Server=" & txtServidorSQL.Text &
";Database=deposito1;UID=" & txtUsuarioSQL.Text & ";PWD=" & txtPassSQL.Text &
";Port=3306;SslMode=none"
            'Se establece la conexión
            Variables.conexion.Open()
            MsgBox("Conectado al servidor", MsgBoxStyle.Information)
            Variables.modoBuffer = False

            'Se muestra el formulario de configuración de la red
            fEnoCAN.Show()

            'Si el sistema estaba funcionando en modo buffer, activamos el envío de
datos a la base
            If Variables.Funcionando = True Then
                fEnoCAN.TimerSQL_Tick(sender, e)
            End If
            Me.Hide()


        Catch ex As Exception
            MsgBox("No se ha podido conectar con el servidor")
        End Try
    End Sub

      End Class
```

Funcionamiento en modo buffer

Esta función permite el funcionamiento del sistema almacenando los valores en el buffer sin introducirlos en la base de datos. Para ello se debe introducir el usuario y contraseña asociados a este modo.

```
Private Sub bInicioBuf_Click(sender As Object, e As EventArgs) Handles
bInicioBuf.Click

        'Si el usuario y contraseña coincide, accedemos al sistema en modo bufer
        If txtUsuarioBuf.Text = "modobuffer" And txtPassBuf.Text = "passmodobuffer"
Then
            Variables.modoBuffer = True
            Me.Hide()
            fINICIO.Hide()
            fEnoCAN.Show()
        Else
            MsgBox("Usuario o contraseña incorrectos", 0, "Acceso denegado")
            txtPassBuf.Text = ""
            txtUsuarioBuf.Text = ""
            Variables.modoBuffer = False
        End If

           End Sub
```

Función ValidaNodos()

Una vez se ha accedido a la ventana de configuración, bien a través de la conexión a la base de datos o bien mediante la conexión en modo buffer, se debe seleccionar el número de nodos mediante una *TextList*. Según el valor que se haya seleccionado aparecerán tantas pestañas asociadas a los nodos, como nodos se hayan seleccionado.

```
Private Sub bSelecNodos_Click(sender As Object, e As EventArgs) Handles
bSelecNodos.Click
        Me.Size = New Size(1085, 554)
        tbEntradas.Visible = True
        ValidaNodos()

           End Sub
```

```
Public Sub ValidaNodos() 'Muestra las pestañas correspondientes según los nodos
seleccionados
        Dim numnodos As Integer = cbNodos.SelectedIndex

        If numnodos >= 0 Then
            tbNodo1.Parent = tbEntradas
        Else
            tbNodo1.Parent = Nothing
        End If

        If numnodos >= 1 Then
            tbNodo2.Parent = tbEntradas
        Else
            tbNodo2.Parent = Nothing
        End If

        If numnodos >= 2 Then
            tbNodo3.Parent = tbEntradas
        Else
            tbNodo3.Parent = Nothing
        End If

              ...

       'ETC. Mismo procedimiento para los 10 nodos
End Sub
```

Añadir/quitar horas de muestra

Cada entrada de cada nodo dispone de una *ListBox* en la cual se guardan las horas a las que se desea que se realice una lectura de la entrada. Con dos botones de añadir y quitar, se modifican estas horas.

```
Private Sub bMasHN1A0_Click(sender As Object, e As EventArgs) Handles
bMasHN1A0.Click
        'Añade a la ListBox la hora introducida en el TextBox
        lbHoraN1A0.Items.Add(txtHoraN1A0.Text)
    End Sub

    Private Sub bQuitaHN1A0_Click(sender As Object, e As EventArgs) Handles
bQuitaHN1A0.Click
        'Si hay horas introducidas, se elimina la última
        If (lbHoraN1A0.SelectedIndex > -1) Then
            lbHoraN1A0.Items.RemoveAt(lbHoraN1A0.SelectedIndex)
        End If

    End Sub
```

Puesta en marcha

Una vez configurados todos los nodos ya puede comenzar el sistema a funcionar. En primer lugar, vacían las variables que nos indican si ya se ha enviado la trama correspondiente a una hora. Esto se hace para que solo se solicite una muestra cada hora establecida.

Después se crean las bases y tablas para cada nodo y entrada. También se abre la conexión serie con el nodo maestro y se habilitan los *Timer*.

```
Private Sub bComenzar_Click(sender As Object, e As EventArgs) Handles
bComenzar.Click
        lbFuncionando.Visible = True

        tbEntradas.Enabled = False

        'Al principio de cada dia se resetea el array que nos indica si se ha
enviado a la trama para esa hora
        Array.Clear(horasN1A, 0, 23)
        Array.Clear(horasN1D7, 0, 23)
        Array.Clear(horasN1S, 0, 23)
        Array.Clear(horasN1B, 0, 23)

        'Creamos el archivo excell a modo de buffer
        ExisteExcel()
        'Creamos las tablas en la base de datos si no están creadas
        CreaTablasSQL()

        If Variables.modoBuffer = False Then
            TimerSQL.Enabled = True
            lbBuffer.Visible = False
        Else
            lbBuffer.Visible = True
            TimerSQL.Enabled = False
        End If



        Try  'Abrimos la conexión del puerto serie
            SerialPort.PortName = txtCOM.Text
            SerialPort.Open()

            TimerCheckHora.Enabled = True
            TimerEnvioSerie.Enabled = True

        Catch ex As Exception
            MsgBox("No se pudo inicializar el puerto serie")
            MsgBox(ex.ToString)
        End Try
    End Sub
```

Crear tabla SQL

Para crear cada base de datos se comprueba el número de nodos seleccionados y se crea una base de datos con el nombre de cada nodo. Después, para cada nodo, se crean las tablas correspondientes a las entradas seleccionadas con el nombre de cada una

```
Public Sub CreaTablasSQL() ' Esta función comprueba si existen las tabas y las
bases creadas en la base de datos y comienza la comunicación serie
        nombrenodo(0) = txtNodo1.Text


        For i As Integer = 0 To cbNodos.SelectedIndex 'Creamos una base de datos
para cada nodo si es que no está creada
            CrearBD(nombrenodo(i))
        Next

        If cbN1A0.Checked = True Then  'Si esta seleccionada la entrada, creamos
una tabla para esa entrada si es que no está creada
            tablaN1A(0) = txtTablaN1A0.Text
            CrearTabla(tablaN1A(0), nombrenodo(0))
        End If

        If cbN1A1.Checked = True Then  'Si esta seleccionada la entrada, creamos
una tabla para esa entrada si es que no está creada
            tablaN1A(1) = txtTablaN1A1.Text
            CrearTabla(tablaN1A(1), nombrenodo(0))
        End If

                            ...


‘ETC. Se realiza este procedimiento para cada entrada seleccionada de cada nodo
End Sub




Public Sub CrearTabla(tabla As String, base As String)

        If Variables.modoBuffer = False Then
            Variables.comando.Connection = Variables.conexion
            Dim sql As String
            Try
                sql = "USE " & base & "; CREATE TABLE " & tabla & "( Fecha datetime
NOT NULL, Valor decimal(9,2) NOT NULL)"
                EjecutarSQL(sql)
            Catch ex As Exception
                MsgBox(“La tabla ya existe")
            End Try
        End If
    End Sub
```

```
    Public Sub CrearBD(base As String)
        If Variables.modoBuffer = False Then
            Variables.comando.Connection = Variables.conexion
            Dim sql As String
            Try
                sql = "CREATE DATABASE " & base
                EjecutarSQL(sql)
            Catch ex As Exception
                MsgBox("La base de datos ya existe")
            End Try
        End If
    End Sub


    Public Sub EjecutarSQL(ByVal SentenciaSQL As String)

        Variables.comando.CommandText = SentenciaSQL

        Variables.comando.ExecuteNonQuery()
    End Sub
```

Chequeo de la hora

Una vez arrancado el sistema, continuamente se comprueba la hora actual con las horas establecidas para la lectura de cada entrada. Como solo se desea que se realice una medida para cada hora establecida, cada vez que se realiza una lectura se almacena en una variable la hora a la que se realiza. De esta forma, cuando la hora actual coincide con la hora de lectura de alguna entrada, se comprueba si esa lectura se ha realizado ya. Cada nuevo día se vacían estas variables.

Si se determina que se debe realizar una lectura, se introduce la trama de solicitud de lectura al buffer donde se introducen las tramas a enviar por puerto serie al nodo maestro. Después, se introduce en la variable la hora actual para que no se vuelva a solicitar la lectura.

```
    Private Sub TimerCheckHora_Tick(sender As Object, e As EventArgs) Handles
TimerCheckHora.Tick

        If Now.ToString("HH:mm") = "00:00" Then  'Solo queremos que para cada hora
se realice una medida
            Array.Clear(horasN1A, 0, 23) 'Al principio de cada dia se resetea la
variable que nos indica si se ha enviado a la trama para esa hora
            Array.Clear(horasN1D7, 0, 23)
            Array.Clear(horasN1S, 0, 23)
            Array.Clear(horasN1B, 0, 23)

        End If
```

```
        If cbN1A0.Checked = True Then   'Si la entrada está seleccionada comprueba
si en la hora actual hay que solicitar una medida

            Dim byteTramaEnvioPS(2) As Byte
            byteTramaEnvioPS = {40, 65, 48} ' (A0 Trama correspondiente al nodo y
entrada
            Dim checksum As Byte = CalculaChecksum(byteTramaEnvioPS)
            Dim tramaEnvioPS As String = Chr(byteTramaEnvioPS(0)) &
Chr(byteTramaEnvioPS(1)) & Chr(byteTramaEnvioPS(2)) & Chr(checksum)

            For i As Integer = 0 To lbHoraN1A0.Items.Count - 1 'Recorremos todas
las horas introducidas y la comparamos con la hora actual
                If lbHoraN1A0.Items(i).ToString = Now.ToString("HH:mm") And
horasN1A(0, i) = 0 Then 'Si la hora actual es una hora seleccionada y ho se ha
enviado ya, solicitamos la información
                    Try
                        env_FIFOPS.Enqueue(tramaEnvioPS)
                        System.Threading.Thread.Sleep(200)
                        horasN1A(0, i) = 1  'Indicamos que esta hora ya se ha
enviado
                    Catch ex As Exception
                        MsgBox(ex.Message)
                    End Try
                End If
            Next
        End If

        If cbN1A1.Checked = True Then   'Si la entrada está seleccionada

            Dim byteTramaEnvioPS(2) As Byte
            byteTramaEnvioPS = {40, 65, 49} ' (A1 Trama correspondiente al nodo y
entrada
            Dim checksum As Byte = CalculaChecksum(byteTramaEnvioPS)
            Dim tramaEnvioPS As String = Chr(byteTramaEnvioPS(0)) &
Chr(byteTramaEnvioPS(1)) & Chr(byteTramaEnvioPS(2)) & Chr(checksum)

            For i As Integer = 0 To lbHoraN1A1.Items.Count - 1 'Recorremos todas
las horas introducidas y la comparamos con la hora actual
                If lbHoraN1A1.Items(i).ToString = Now.ToString("HH:mm") And
horasN1A(1, i) = 0 Then 'Si la hora actual es una hora seleccionada, solicitamos la
información
                    Try
                        env_FIFOPS.Enqueue(tramaEnvioPS)
                        System.Threading.Thread.Sleep(200)
                        horasN1A(1, i) = 1
                    Catch ex As Exception
                        MsgBox(ex.Message)
                    End Try
                End If
            Next

        End If

                                    ...
'De igual manera para el resto de entradas
End Sub
```

Recibir datos por puerto serie

Este evento se desencadena cada vez que se reciben datos por el puerto serie. Se almacena en una variable los datos recibidos y se comprueba uno a uno el valor de esos datos. Como la trama recibida tiene que finalizar con el carácter '#', cuando se detecta que se ha recibido ese carácter se introduce la trama en el archivo Excel a modo de buffer. Según en que modo de operación se encuentre el sistema, se activa o no el *Timer* mediante el cual se van introduciendo los registros del buffer en la base de datos.

```
Private Sub SerialPort_DataReceived(sender As Object, e As
IO.Ports.SerialDataReceivedEventArgs) Handles SerialPort.DataReceived

        If pausa = False Then
            Try

                buffer = SerialPort.ReadExisting
                cadena = cadena + buffer
                Dim tam As Integer = Len(buffer)
                For indice As Integer = 0 To (tam - 1)
                    If buffer(indice) = "#" Then
                        'Si se recibe # quiere decir que la cadena ha terminado
                        rec_FIFOPS.Enqueue(cadena)
                        Me.Invoke(New EventHandler(AddressOf GuardaDatos))
                    End If
                Next

            Catch ex As Exception
                MsgBox(ex.Message)
            End Try
        End If
    End Sub

    Public Sub GuardaDatos(ByVal sender As Object, ByVal e As System.EventArgs)
'Subrutina a parte ya que no deja desde la del PS
        cadena = ""
        tramaRecibidaPS = rec_FIFOPS.Dequeue()
        'Se introducen en el buffer excel
        IntroducirExcel(tramaRecibidaPS)

        Variables.Libro.Save()

        If Variables.modoBuffer = False Then
            TimerSQL.Enabled = True
            lbBuffer.Visible = False
        Else
            TimerSQL.Enabled = False
            lbBuffer.Visible = True
        End If
    End Sub
```

Cálculo del Checksum

Esta función calcula el CS de una cadena dada. Se emplea para calcular el CS de la trama que se envía al nodo maestro vía puerto serie para solicitar la lectura de alguna entrada. Calcula la 'or exclusiva' de la cadena. De esta forma, cuando el nodo maestro calcule el checksum de toda la cadena recibida, incluido el CS, deberá obtener un cero como resultado si los datos se han recibido de forma íntegra.

```
Function CalculaChecksum(byteTrama() As Byte) As Byte
        Dim checksum As Byte = 0
        For i As Integer = 0 To byteTrama.Length - 1
            checksum = checksum Xor byteTrama(i)
        Next
        Return checksum
    End Function
```

Creación del archivo Excel a modo de buffer

Esta función crea el archivo Excel. Configura las 4 columnas que va a ocupar cada registro como celdas de tipo texto. Además, introduce el carácter '#' que se emplea para calcular cuántos registros hay introducidos.

```
Public Sub ExisteExcel()  'Crea el buffer en Excell si es que no está creado

        If System.IO.File.Exists("C:\Users\usuario\Desktop\buffer.xlsx") Then

            Variables.Libro =
Variables.ApExcel.Workbooks.Open("C:\Users\usuario\Desktop\buffer.xlsx")
            Variables.Libro.Worksheets("Hoja1").Columns("A").Numberformat = "@"
            Variables.Libro.Worksheets("Hoja1").Columns("B").Numberformat = "@"
            Variables.Libro.Worksheets("Hoja1").Columns("C").Numberformat = "@"
            Variables.Libro.Worksheets("Hoja1").Columns("D").Numberformat = "@"
        Else
            Variables.Libro = Variables.ApExcel.Workbooks.Add

Variables.Libro.SaveAs(Filename:="C:\Users\usuario\Desktop\buffer.xlsx")
            Variables.Libro =
Variables.ApExcel.Workbooks.Open("C:\Users\usuario\Desktop\buffer.xlsx")
            Variables.Libro.Worksheets("Hoja1").Columns("A").Numberformat = "@"
            Variables.Libro.Worksheets("Hoja1").Columns("B").Numberformat = "@"
            Variables.Libro.Worksheets("Hoja1").Columns("C").Numberformat = "@"
            Variables.Libro.Worksheets("Hoja1").Columns("D").Numberformat = "@"
            Variables.Libro.Worksheets("Hoja1").Cells(1, 1) = "#"
        End If
    End Sub
```

Introducir datos al buffer Excel

Esta función introduce las tramas recibidas por el puerto serie al archivo Excel. En primer lugar, comprueba que la trama que se recibe sea una trama de datos, y no una trama de error. Si es una trama de error se vuelve a enviar la última solicitud de datos. Si no, se descompone la trama y se introduce cada valor en su columna correspondiente en el archivo Excel. Se introduce nombre del nodo, nombre de la variable, valor de la variable y hora de la lectura.

Como el valor de la lectura recibida será un entero entre 0-255, en esta función se aplica los valores configurados en la recta de conversión para obtener el valor real de la lectura.

```
Public Sub IntroducirExcel(cadena As String)

        Dim divTrama() As String = Split(tramaRecibidaPS, ";") 'Separamos la trama
de en caracteres individuales. Estará en formato
Nodo(d);Canal(d);Entrada(d);Valor(d);Cs(d);#

        txtSeparar.Text = divTrama(0) & ";" & Chr(divTrama(1)) & ";" &
Chr(divTrama(2)) & ";" & divTrama(3) & ";" & divTrama(4) & ";" & divTrama(5)

        If CInt(divTrama(0)) = 1 Then  'Si el nodo maestro nos envia el mensaje de
error, intentamos la conexión tres veces
            If intentos < 3 Then
                TimerEnvioSerie.Enabled = False
                SerialPort.Write(ultimaTramaEnviadaPS)
                intentos = intentos + 1
            Else
                MsgBox("Error de conexión serie.", MsgBoxStyle.Critical)
            End If

        Else
            TimerEnvioSerie.Enabled = True
            intentos = 0
            Select Case CInt(divTrama(0)) 'Offset de la dirección del nodo -> Nodo
0 = 40, Nodo 1 = 41
                'En función de la direccion del nodo, se comprueba de que nodo
procede la entrada
                Case 40
                    Variables.depositoExc = nombrenodo(0)
                Case 41
                    Variables.depositoExc = nombrenodo(1)
                Case 42
                    Variables.depositoExc = nombrenodo(2)
                Case 43
                    Variables.depositoExc = nombrenodo(3)

                                ...

                Case Else

            End Select
```

```
            Dim numEntrada As Integer = divTrama(2) - 48 'Numero de la entrada
analógica, digital, serie o byte. ("0" = 48d)

            Select Case Chr(divTrama(1)) 'Canal de la entrada
                'Guardamos el número de entrada y su valor ya escalado por la recta
de conversión
                Case "A"
                    Variables.variableExc = tablaN1A(numEntrada)
                    Variables.valorExc = aN1A(numEntrada) * CInt(divTrama(3)) +
bN1A(numEntrada)

                Case "D"
                    Variables.variableExc = tablaN1D7(numEntrada)
                    Variables.valorExc = divTrama(3).ToString()
                Case "B"
                    Variables.variableExc = tablaN1B(numEntrada)
                    Variables.valorExc = aN1B(numEntrada) * CInt(divTrama(3)) +
bN1B(numEntrada)
                Case "S"
                    Variables.variableExc = tablaN1S(numEntrada)
                    Variables.valorExc = aN1S(numEntrada) * CInt(divTrama(3)) +
bN1S(numEntrada)
                Case Else

            End Select



            Variables.puntExcel = nReg(Variables.Libro.Worksheets("Hoja1"), 1, 1)
            Variables.Libro.Worksheets("Hoja1").Cells(puntExcel, 1) = depositoExc
            Variables.Libro.Worksheets("Hoja1").Cells(puntExcel, 2) = variableExc
            'VB trabaja con decimales ',' y SQL con '.' Para poder guardarlos en la
base de datos, si el numero contiene decimales, separamos la parte entera de la
decimal y lo introducimos con '.'
            If (valorExc Mod 1) = 0 Then 'Dividimos entre 1 y comprobamos si tiene
resto. Asi sabemos si se trata de un decimal
                Variables.Libro.Worksheets("Hoja1").Cells(puntExcel, 3) = valorExc
            Else
                Dim s_valorExc() As String
                s_valorExc = Split(valorExc.ToString(), ",")
                Variables.Libro.Worksheets("Hoja1").Cells(puntExcel, 3) =
s_valorExc(0) & "." & s_valorExc(1) 'Lo introducimos a Excell con '.' ya que SQL
trabaja con '.' para los decimales
            End If

            Variables.Libro.Worksheets("Hoja1").Cells(puntExcel, 4) =
DateTime.Now.ToString("dd/MM/yyyy HH:mm ")


                Variables.Libro.Worksheets("Hoja1").Cells(Variables.puntExcel + 1,
1) = "#"
            End If

    End Sub
```

Función nReg()

Esta función devuelve el número de fila del archivo Excel donde se encuentra el carácter puntero '#'. En esta fila se introducirá el nuevo registro y el puntero pasará a estar en la fila siguiente.

```
    Public Function nReg(Hoja As Microsoft.Office.Interop.Excel.Worksheet, nFila As
Long, nColumna As Long)
        Do Until Hoja.Cells(nFila, nColumna).Value = "#"
            nFila = nFila + 1
        Loop
        Return nFila
    End Function
```

Introducir datos a la base de datos

A través de este procedimiento se introduce cada registro de datos guardado en el buffer y se introduce a la base de datos. La función ExcelToSQL() es la encargada de tomar el primer registro del buffer y guardarlo en la tabla de la base correspondiente. A esta función se le llama a través de un *Timer*. De esta manera mientras haya datos almacenados en el archivo Excel, se irán introduciendo progresivamente.

En primer lugar, se extrae el valor de cada columna del registro y se crea el comando MySQL con los datos extraídos. Una vez almacenado el dato en la base, se elimina el registro del buffer y se desplaza el resto para proceder a almacenar el siguiente. Si por algún motivo este procedimiento falla, se cancela la transferencia de datos a la base y se siguen almacenando en el buffer Excel.

```
Public Sub ExcelToSQL()
        Variables.comando.Connection = Variables.conexion

        Dim sql As String 'Comando MySQL para introduir los datos

        Try
        Do While Variables.Libro.Worksheets("Hoja1").Cells(1, 1).Value <> "#" 'Se
realiza este procedimiento mientras existan registros

           'Se extrae el valor de cada columna del registro
     Dim valor As String = Variables.Libro.Worksheets("Hoja1").Cells(1, 3).Text
     Dim fechahora As String = Variables.Libro.Worksheets("Hoja1").Cells(1, 4).Text
     Dim dep As String = Variables.Libro.Worksheets("Hoja1").Cells(1, 1).Text ''
     Dim var As String = Variables.Libro.Worksheets("Hoja1").Cells(1, 2).Text ''
                'Creamos el comando
                sql = "use " & dep & "; insert into " & var & " values
(STR_TO_DATE('" & fechahora & "', '%d/%m/%Y %H:%i:%s'), " & valor & ")"
```

```
                'Se ejecuta el envío del comando
                 EjecutarSQL Variables.puntExcel =
 nReg(Variables.Libro.Worksheets("Hoja1"), 1, 1)

              For i As Integer = 1 To (puntExcel - 1)
              'Se elimina el registro ya introducido a la base y desplaza el resto
                    Variables.Libro.Worksheets("Hoja1").Cells(i, 1).Value =
 Variables.Libro.Worksheets("Hoja1").Cells(i + 1, 1).Value

                     Variables.Libro.Worksheets("Hoja1").Cells(i, 2).Value =
Variables.Libro.Worksheets("Hoja1").Cells(i + 1, 2).Value

                     Variables.Libro.Worksheets("Hoja1").Cells(i, 3).Value =
 Variables.Libro.Worksheets("Hoja1").Cells(i + 1, 3).Value

                     Variables.Libro.Worksheets("Hoja1").Cells(i, 4).Value =
 Variables.Libro.Worksheets("Hoja1").Cells(i + 1, 4).Value
                 Next
             Variables.Libro.Worksheets("Hoja1").Cells(Variables.puntExcel, 1) = ""
             Variables.Libro.Worksheets("Hoja1").Cells(Variables.puntExcel, 2) = ""
             Variables.Libro.Worksheets("Hoja1").Cells(Variables.puntExcel, 3) = ""
             Variables.Libro.Worksheets("Hoja1").Cells(Variables.puntExcel, 4) = ""
             Loop

         Catch ex As Exception
             'Si no se pueden introducir a la base, se activa el modo buffer
             MsgBox(ex.ToString)
             MsgBox("Error. Se ha activado el modo buffer")
             modoBuffer = True

         End Try
     End Sub


 'Timer que desencadena la transferencia de datos del Excel a la base de datos
     Public Sub TimerSQL_Tick(sender As Object, e As EventArgs) Handles
 TimerSQL.Tick
         ExcelToSQL()
         TimerSQL.Enabled = False

     End Sub
```

Timer de envío por puerto serie

Cuando el sistema está funcionando, se extraen progresivamente las tramas almacenadas en el buffer FIFO donde se van introduciendo las tramas que se desean enviar por puerto serie. Estas tramas corresponden con la solicitud de la lectura de las entradas de los nodos. Mientras haya tramas pendientes de enviar, se extraen y se envían por el puerto serie al nodo maestro progresivamente.

```
Private Sub TimerEnvioSerie_Tick(sender As Object, e As EventArgs) Handles
TimerEnvioSerie.Tick  'Cada segundo envia por puerto serie las cadenas guardadas en
la memoria FIFO
        Try
            Dim tramaEnvioPS As String = env_FIFOPS.Dequeue()
            ultimaTramaEnviadaPS = tramaEnvioPS
            SerialPort.Write(tramaEnvioPS)
        Catch ex As Exception
            'Si no hay nada endiente de envío que no haga nada
        End Try
    End Sub
```

Representar los valores de una tabla MySQL

A través de esta función, se representan en la aplicación los valores almacenados de una entrada concreta. Para ello es necesario introducir el nombre de la entrada y el nodo al que pertenece. Con ello se crea el comando para extraer los datos de la base y representarlos en un *DataGrid.*

```
Private Sub bCargar_Click(sender As Object, e As EventArgs) Handles bCargar.Click
        dgParametros.Visible = True

        If txtNombreTabla.Text <> "" And txtNombreBase.Text <> "" Then 'Obtenemos
el nombre de la base y tabla a consultar
            nombretabla = txtNombreTabla.Text
            nombrebase = txtNombreBase.Text
        End If
        VisualizarSQL(nombrebase, nombretabla)
    End Sub

    Public Sub VisualizarSQL(ByVal base As String, ByVal tabla As String)
        Try
            Dim consulta As String

            consulta = "use " & base & " ; select * from " & tabla
            Variables.adaptador = New MySqlDataAdapter(consulta,
Variables.conexion)
            Variables.datos = New DataSet
            Variables.adaptador.Fill(Variables.datos, tabla)
            dgParametros.DataSource = Variables.datos
            dgParametros.DataMember = tabla
        Catch ex As Exception
            MsgBox(ex.ToString())
            MsgBox("No se pudo encontrar la base de datos.")
        End Try

    End Sub
```

Representar un gráfico a partir de una tabla MySQL

Esta función muestra de forma gráfica los datos representados en la tabla.

```
    Private Sub bGrafico_Click(sender As Object, e As EventArgs) Handles
bGrafico.Click
        cGrafico.Visible = True
        verGrafico()
    End Sub

    Public Sub verGrafico()
        cGrafico.Titles.Clear()
        cGrafico.Series(0).LegendText = "Valor"
        cGrafico.Titles.Add(txtNombreTabla.Text)

        Dim trama As New MySqlCommand("select * from " & nombretabla,
Variables.conexion)
        Dim reader As MySqlDataReader = trama.ExecuteReader
        cGrafico.Series(0).Points.DataBindXY(reader, txtEjeX.Text, reader,
txtEjeY.Text)

        cGrafico.Series(0).ChartType =
DataVisualization.Charting.SeriesChartType.Spline

    End Sub
```

3.5 LIBRERÍAS EMPLEADAS

En este anexo se va a presentar la librería ChipKITCAN. Esta librería está contenida en el entorno de desarrollo MPIDE para facilitar la programación de aplicaciones que empleen los controladores CAN que presenta la placa.

```
/************************************************************************/
/*

/*        chipKITCan.h    --  Interface Declarations for chipKITCan.cpp       */
/*

/************************************************************************/
/*        Author:     Gene Apperson
          Copyright (c) 2011, Digilent Inc. All rights reserved.

         Traducción: Daniel Pérez García                                  */
/************************************************************************/

#if !defined(CHIPKIT_CAN_H)

 /* La clase CAN declara el objeto empleado para acceder a los controladores CAN
     en el microcontrolador PIC32MX795
      Ejemplo:
  CAN     can1(CAN::CAN1);    //CAN object accessing CAN module 1
  CAN::CHANNEL        chn;    //declaración de una variable canal CAN
  CAN::CHANNEL_EVENT  evt;    //declaración de una variable de evento del canal
  chn = CAN::CHANNEL0;        //asignación del número de canal a una variable
 evt = can1.getChannelEvent(CAN::CHANNEL7);  //introduce el numero del canal como parámetro

*/


class CAN {

public:

/* ------------------------------------------------------------ */
/*         DECLARACIÓN DE OBJETOS DE LA CLASE CAN               */
/* ------------------------------------------------------------ */

typedef enum {
          FALSE = 0,
          TRUE
} BOOL;

/* Identificador del módulo CAN
*/
typedef enum {
    CAN1,
    CAN2,
    NUM_CAN_MODULES,
    MOD_NIL = 0xFF
} MODULE;

/* ------------------------------------------------------------ */
/* Identificador del canal CAN.
*/
typedef enum {
          CHANNEL0,       // Canal 0 ID
          CHANNEL1,       // Canal 1 ID
          CHANNEL2,       // Canal 2 ID
          CHANNEL3,       // Canal 3 ID
          CHANNEL4,       // Canal 4 ID
          CHANNEL5,       // Canal 5 ID
          CHANNEL6,       // Canal 6 ID
          CHANNEL7,       // Canal 7 ID
          CHANNEL8,       // Canal 8 ID
          CHANNEL9,       // Canal 9 ID
          CHANNEL10,      // Canal 10 ID
```

```
        CHANNEL11,      // Canal 11 ID
        CHANNEL12,      // Canal 12 ID
        CHANNEL13,      // Canal 13 ID
        CHANNEL14,      // Canal 14 ID
        CHANNEL15,      // Canal 15 ID
        CHANNEL16,      // Canal 16 ID
        CHANNEL17,      // Canal 17 ID
        CHANNEL18,      // Canal 18 ID
        CHANNEL19,      // Canal 19 ID
        CHANNEL20,      // Canal 20 ID
        CHANNEL21,      // Canal 21 ID
        CHANNEL22,      // Canal 22 ID
        CHANNEL23,      // Canal 23 ID
        CHANNEL24,      // Canal 24 ID
        CHANNEL25,      // Canal 25 ID
        CHANNEL26,      // Canal 26 ID
        CHANNEL27,      // Canal 27 ID
        CHANNEL28,      // Canal 28 ID
        CHANNEL29,      // Canal 29 ID
        CHANNEL30,      // Canal 30 ID
        CHANNEL31,      // Canal 31 ID
        ALL_CHANNELS    // unicamente empleado en la función CANAbortPendingTx() .
} CHANNEL;

/* ------------------------------------------------------------ */
/* Modos de operación del módulo CAN:

**  NORMAL_OPERATION    - Modo normal. El módulo transmite y recibe mensajes.
**  DISABLE             - Modulo deshabilitado. El módulo ni transmite ni recibe mensajes
**  LOOPBACK            - En este modo, el TX está internamente conectado al RX.
**  LISTEN_ONLY         - Modo escucha. El modulo captura todos los mensajes pero no actúa.
**  CONFIGURATION       - Los ajustes del módulo se configuran en este modo.
**  LISTEN_ALL_MESSAGES - Escucha todos los mensajes independientemente de los errores.
*/
typedef enum {
        NORMAL_OPERATION,
        DISABLE,
        LOOPBACK,
        LISTEN_ONLY,
        CONFIGURATION,
        LISTEN_ALL_MESSAGES = 7
} OP_MODE;

/* ------------------------------------------------------------ */
/* Eventos del canal CAN:

  Ennumera todos los eventos del canal TX y RX. Los miembros de esta
  lista se pueden emplear para habilitar o deshabilitar los eventos del canal
  o como una máscara para comprobar si el evento del canal está habilitado
*/
typedef enum {
        RX_CHANNEL_NOT_EMPTY        = 0x1,
        RX_CHANNEL_HALF_FULL        = 0x2,
        RX_CHANNEL_FULL    = 0x4,
        RX_CHANNEL_OVERFLOW         = 0x8,
        RX_CHANNEL_ANY_EVENT        = 0xF,
        TX_CHANNEL_EMPTY   = 0x100,
        TX_CHANNEL_HALF_EMPTY       = 0x200,
        TX_CHANNEL_NOT_FULL= 0x400,
        TX_CHANNEL_ANY_EVENT        = 0x700
} CHANNEL_EVENT;

/* ------------------------------------------------------------ */
/* Bits del TQ:

  Esta lista obtiene el valore que se puede emplear para
  definir el numero de TQ por bit
*/
typedef enum {
        BIT_1TQ,    // 1-bit Time Quanta
        BIT_2TQ,    // 2-bit Time Quanta
        BIT_3TQ,    // 3-bit Time Quanta
        BIT_4TQ,    // 4-bit Time Quanta
```

```
        BIT_5TQ,    // 5-bit Time Quanta
        BIT_6TQ,    // 6-bit Time Quanta
        BIT_7TQ,    // 7-bit Time Quanta
        BIT_8TQ     // 8-bit Time Quanta
} BIT_TQ;

/* ------------------------------------------------------------ */
/* Configuración de los Bits del CAN
*/
typedef struct
{
        BIT_TQ    phaseSeg2Tq;
        BIT_TQ    phaseSeg1Tq;
        BIT_TQ    propagationSegTq;
        BOOL      phaseSeg2TimeSelect;
        BOOL      sample3Time;
        BIT_TQ    syncJumpWidth;
} BIT_CONFIG;

/* ------------------------------------------------------------ */
/* Codigo de eventos del CAN
** Se obtiene el código del evento devuelto por getPendingEventCode
*/
typedef enum {
        CHANNEL0_EVENT,     // Hay activo un evento ene l Canal 0
        CHANNEL1_EVENT,
        CHANNEL2_EVENT,
        CHANNEL3_EVENT,
        CHANNEL4_EVENT,
        CHANNEL5_EVENT,
        CHANNEL6_EVENT,
        CHANNEL7_EVENT,
        CHANNEL8_EVENT,
        CHANNEL9_EVENT,
        CHANNEL10_EVENT,
        CHANNEL11_EVENT,
        CHANNEL12_EVENT,
        CHANNEL13_EVENT,
        CHANNEL14_EVENT,
        CHANNEL15_EVENT,
        CHANNEL16_EVENT,
        CHANNEL17_EVENT,
        CHANNEL18_EVENT,
        CHANNEL19_EVENT,
        CHANNEL20_EVENT,
        CHANNEL21_EVENT,
        CHANNEL22_EVENT,
        CHANNEL23_EVENT,
        CHANNEL24_EVENT,
        CHANNEL25_EVENT,
        CHANNEL26_EVENT,
        CHANNEL27_EVENT,
        CHANNEL28_EVENT,
        CHANNEL29_EVENT,
        CHANNEL30_EVENT,
        CHANNEL31_EVENT,
        NO_EVENT = 0x40,    // No hay eventos activos
        ERROR_EVENT,        // Evento de error en el CAN bus activo.
        WAKEUP_EVENT,       // Evento de inicialización del CAN activo.
        RX_CHANNEL_OVERFLOW_EVENT,  // Evento de overflow en la recepción activo.
        ADDRESS_ERROR_EVENT,        // Evento de error en la dirección activo.
        BUS_BANDWIDTH_ERROR,        // Evento de error de ancho de banda activo.
        TIMESTAMP_TIMER_EVENT,      // Evento de overflow del timer Timestamp activo.
        MODE_CHANGE_EVENT,          // Evento de cambio de módulo activo.
        INVALID_MESSAGE_RECEIVED_EVENT  // Evento de mensaje recibido no válido activo.
} EVENT_CODE;


/* ------------------------------------------------------------ */
/* Identificadores de los filtros can
*/
typedef enum {
        FILTER0,
```

```
          FILTER1,
          FILTER2,
          FILTER3,
          FILTER4,
          FILTER5,
          FILTER6,
          FILTER7,
          FILTER8,
          FILTER9,
          FILTER10,
          FILTER11,
          FILTER12,
          FILTER13,
          FILTER14,
          FILTER15,
          FILTER16,
          FILTER17,
          FILTER18,
          FILTER19,
          FILTER20,
          FILTER21,
          FILTER22,
          FILTER23,
          FILTER24,
          FILTER25,
          FILTER26,
          FILTER27,
          FILTER28,
          FILTER29,
          FILTER30,
          FILTER31,

    /* Numero total de filtros del módulo
    */
    NUMBER_OF_FILTERS
} FILTER;

/* ------------------------------------------------------------ */
/* Mascaras de los filtros CAN
*/
typedef enum {
          FILTER_MASK0,
          FILTER_MASK1,
          FILTER_MASK2,
          FILTER_MASK3,

    // Número total de mascaras de filtros
    NUMBER_OF_FILTER_MASKS
} FILTER_MASK;

/* ------------------------------------------------------------ */
/* Tipo de identificador CAN

*/
typedef enum {
          EID,     // Extendido -> 29 bits
          SID      // Estándar -> 11 bits
} ID_TYPE;

/* ------------------------------------------------------------ */
/* Petición de transmisión remota (RTR)
*/
typedef enum {
          TX_RTR_ENABLED,
          TX_RTR_DISABLED   /
} TX_RTR;

/* ------------------------------------------------------------ */
/* Modo de recepción única de datos

*/
typedef enum {
          RX_DATA_ONLY,
```

```
        RX_FULL_RECEIVE
} RX_DATA_MODE;

/* ----------------------------------------------------------- */
/* Tipos de máscaras
*/
typedef enum {
        FILTER_MASK_IDE_TYPE,
        FILTER_MASK_ANY_TYPE
} FILTER_MASK_TYPE;

/* ----------------------------------------------------------- */
/* Prioridad de transmisiónd el canal
*/
typedef enum {
        LOWEST_PRIORITY,
        LOW_MEDIUM_PRIORITY,
        HIGH_MEDIUM_PRIORITY,
        HIGHEST_PRIORITY
} TXCHANNEL_PRIORITY;

/* ----------------------------------------------------------- */
/* Tamañ de la red de filtros
*/
typedef enum {
        DNET_FILTER_DISABLE,
        DNET_FILTER_SIZE_1_BIT,
    DNET_FILTER_SIZE_2_BIT,
    DNET_FILTER_SIZE_3_BIT,
        DNET_FILTER_SIZE_4_BIT,
        DNET_FILTER_SIZE_5_BIT,
        DNET_FILTER_SIZE_6_BIT,
        DNET_FILTER_SIZE_7_BIT,
        DNET_FILTER_SIZE_8_BIT,
        DNET_FILTER_SIZE_9_BIT,
        DNET_FILTER_SIZE_10_BIT,
        DNET_FILTER_SIZE_11_BIT,
        DNET_FILTER_SIZE_12_BIT,
        DNET_FILTER_SIZE_13_BIT,
        DNET_FILTER_SIZE_14_BIT,
        DNET_FILTER_SIZE_15_BIT,
        DNET_FILTER_SIZE_16_BIT,
        DNET_FILTER_SIZE_17_BIT,
        DNET_FILTER_SIZE_18_BIT,
} DNET_FILTER_SIZE;

/*Eventos del modulo CAN*/
typedef enum {
         RX_EVENT = 0x2,
         TIMESTAMP_TIMER_OVERFLOW_EVEN = 0x4,
         OPERATION_MODE_CHANGE_EVENT = 0x800,
         INVALID_RX_MESSAGE_EVENT = 0x8000
         SYSTEM_ERROR_EVENT = 0x1000,
         BUS_ERROR_EVENT = 0x2000,
         BUS_ACTIVITY_WAKEUP_EVENT = 0x4000,
         INVALID_RX_MESSAGE_EVENT = 0x8000
} MODULE_EVENT;

/* ----------------------------------------------------------- */
/* Estados de error*/
typedef enum {
        TX_RX_WARNING_STATE          = 0x10000,
        RX_WARNING_STATE             = 0x20000,
        TX_WARNING_STATE             = 0x40000,
        RX_BUS_PASSIVE_STATE         = 0x80000,
    TX_BUS_PASSIVE_STATE     = 0x100000,
        TX_BUS_OFF_STATE             = 0x200000
} ERROR_STATE;

/* ----------------------------------------------------------- */
/*Características del módulo*/

typedef enum {
```

```
        STOP_IN_IDLE = 0x2000,
        RX_TIMESTAMP = 0x100000,
        WAKEUP_BUS_FILTER = 0x400000
} MODULE_FEATURES;

/* ------------------------------------------------------------ */
/* Máscaras de los canales CAN
*/
typedef enum {
        CHANNEL0_MASK  = 0x00000001,
        CHANNEL1_MASK  = 0x00000002,
        CHANNEL2_MASK  = 0x00000004,
        CHANNEL3_MASK  = 0x00000008,
        CHANNEL4_MASK  = 0x00000010,
        CHANNEL5_MASK  = 0x00000020,
        CHANNEL6_MASK  = 0x00000040,
        CHANNEL7_MASK  = 0x00000080,
        CHANNEL8_MASK  = 0x00000100,
        CHANNEL9_MASK  = 0x00000200,
        CHANNEL10_MASK = 0x00000400,
        CHANNEL11_MASK = 0x00000800,
        CHANNEL12_MASK = 0x00001000,
        CHANNEL13_MASK = 0x00002000,
        CHANNEL14_MASK = 0x00004000,
        CHANNEL15_MASK = 0x00008000,
        CHANNEL16_MASK = 0x00010000,
        CHANNEL17_MASK = 0x00020000,
        CHANNEL18_MASK = 0x00040000,
        CHANNEL19_MASK = 0x00080000,
        CHANNEL20_MASK = 0x00100000,
        CHANNEL21_MASK = 0x00200000,
        CHANNEL22_MASK = 0x00400000,
        CHANNEL23_MASK = 0x00800000,
        CHANNEL24_MASK = 0x01000000,
        CHANNEL25_MASK = 0x02000000,
        CHANNEL26_MASK = 0x04000000,
        CHANNEL27_MASK = 0x08000000,
        CHANNEL28_MASK = 0x10000000,
        CHANNEL29_MASK = 0x20000000,
        CHANNEL30_MASK = 0x40000000,
        CHANNEL31_MASK = 0x80000000,
        ANYCHANNEL_MASK = 0xFFFFFFFF
} CHANNEL_MASK;

/* ------------------------------------------------------------ */
/* Condiciones del canal TX
*/
typedef enum {
        TX_CHANNEL_TRANSMITTING = 0x8,
        TX_CHANNEL_ERROR        = 0x10,
        TX_CHANNEL_ARBITRATION_LOST = 0x20
} TX_CHANNEL_CONDITION;

/* ------------------------------------------------------------ */
/* Identificación estándar de la tranmisión de mensajes

*/
typedef  struct
{
        unsigned SID:11;
        unsigned :21;
} TX_MSG_SID;

/* ------------------------------------------------------------ */
/* Estructura de los mensajes TX y RX

*/
typedef struct
{
        unsigned DLC:4;
        unsigned RB0:1;
        unsigned :3;
        unsigned RB1:1;
```

```
        unsigned RTR:1;
        unsigned EID:18;
        unsigned IDE:1;
        unsigned SRR:1;
        unsigned :2;
} MSG_EID;

/* ------------------------------------------------------------ */
/* Buffer de transmisión
*/
typedef union {
        struct {
                TX_MSG_SID  msgSID;
                MSG_EID     msgEID;
                uint8_t     data[8];
        };
        uint32_t    messageWord[4];

} TxMessageBuffer;

/* ------------------------------------------------------------ */
/* Identificación del mensaje de transmisión*/

typedef   struct {
        unsigned SID:11;    /
        unsigned FILHIT:5;  /
        unsigned CMSGTS:16; /

} RX_MSG_SID;

/* ------------------------------------------------------------ */
/* Buffer de recepción de mensajes
*/
typedef union {
    /* This structure is used for a full message.
    */
        struct {
                RX_MSG_SID  msgSID;
        };

    uint8_t     dataOnlyMsgData[8];
    uint32_t    messageWord[4];

} RxMessageBuffer;

/* ------------------------------------------------------------ */
/*          Variables de la clase CAN                           */
/* ------------------------------------------------------------ */
private:
    MODULE  mod;


/* ------------------------------------------------------------ */
/*          Funciones de la clase CAN                         */
/* ------------------------------------------------------------ */
public:
            CAN(MODULE modNew);
           ~CAN();

    /* Métodos de interrupción
    */
    void        attachInterrupt(void (*pfn)());
    void        detachInterrupt();

    /* Métodos de configuración
    */
    void        assignMemoryBuffer(void * buf, uint32_t size);
    void        setOperatingMode(OP_MODE opm);
    OP_MODE     getOperatingMode();
    void        enableFeature(MODULE_FEATURES features, bool enable);
    void        deviceNetFilter(DNET_FILTER_SIZE dncnt);
    void        setTimeStampValue(uint32_t val);
    uint32_t    getTimeStampValue();
```

```
    void        setTimeStampPrescalar(uint32_t pre);
    void        enableModule(bool enable);
    void        setSpeed(const BIT_CONFIG * cfg, uint32_t clk, uint32_t spd);
    bool        isActive();
    void        resetChannel(CHANNEL chn);
    bool        isChannelReset(CHANNEL chn);
    void        updateChannel(CHANNEL chn);

    /* Métodos de manejo de eventos
    */
    void        enableModuleEvent(MODULE_EVENT evt, bool enable);
    MODULE_EVENT getModuleEvent();
    void        clearModuleEvent(MODULE_EVENT evt);
    void        enableChannelEvent(CHANNEL chn, CHANNEL_EVENT evt, bool enable);
    EVENT_CODE  getPendingEventCode();
    CHANNEL_MASK getAllChannelEventStatus();
    CHANNEL_MASK getAllChannelOverflowStatus();
    CHANNEL_EVENT getChannelEvent(CHANNEL chn);
    void        clearChannelEvent(CHANNEL chn, CHANNEL_EVENT evt);

    /* Funciones de transmisión de mensajes
    */
    void        configureChannelForTx(CHANNEL chn, uint32_t size, TX_RTR rtren,
TXCHANNEL_PRIORITY pr);
    void        abortPendingTx(CHANNEL chn);
    void        flushTxChannel(CHANNEL chn);
    TX_CHANNEL_CONDITION getTxChannelCondition(CHANNEL chn);
    TxMessageBuffer * getTxMessageBuffer(CHANNEL chn);
    bool        isTxAborted(CHANNEL chn);

    /* Funciones de recepción de mensajes
    */
    void        configureChannelForRx(CHANNEL chn, uint32_t size, RX_DATA_MODE dataOnly);
    RxMessageBuffer * getRxMessage(CHANNEL chn);

    /* Funciones de filtrado de mensajes
    */
    void        configureFilterMask(FILTER_MASK mask, uint32_t maskbits, ID_TYPE id,
FILTER_MASK_TYPE mide);
    void        configureFilter(FILTER filter, uint32_t id, ID_TYPE type);
    void        enableFilter(FILTER filter, bool enable);
    FILTER      getLatestFilterHit();
    void        linkFilterToChannel(FILTER filter, FILTER_MASK mask, CHANNEL chn);
    bool        isFilterDisabled(FILTER filter);

    /* Trazado de estados de error
    */
    uint32_t    getRxErrorCount();
    uint32_t    getTxErrorCount();
    ERROR_STATE getErrorState();

    /* Funciones de información
    */
    uint32_t    totalModules();
    uint32_t    totalChannels();
    uint32_t    totalFilters();
    uint32_t    totalMasks();
};


#endif

/******************************************************************************/
```

3.6 CODIGO DEL FIRMWARE

```
/*
PROYECTO FINAL DE CARRERA

Autor: Daniel Pérez García
Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Universidad de La Rioja

Septiembre 2018

*/

#include  <WProgram.h>
#include  "chipKITCAN.h"
#include <LiquidCrystal.h>

#define SYS_FREQ   (80000000L)
#define CAN_BUS_SPEED   250000                  // CAN Speed

CAN    canMod1(CAN::CAN1);    // this object uses CAN module 1
uint8_t  CAN1MessageFifoArea[2 * 8 * 16];
static volatile bool isCAN1MsgReceived = false;

//FUNCIONES GENERALES
void initCan1(uint32_t myaddr); // Inicializa el controlador can
void doCan1Interrupt(); // Habilita interrupciones
int IDnodo(); // Lee la entrada del switch que define la dirección del nodo

//VARIABLES GENERALES

int nodocan; // Direccion del nodo en el que trabajará la placa
bool nodoMaestro = false;


//FUNCIONES DEL NODO MAESTRO
void recibePS(); // Obtiene la trama recibida via puerto serie
void txCAN_Maestro(uint32_t rxnode); // Envio del nodo Maestro
void rxCAN_Maestro(void); // Recepción del nodo Maestro

//VARIABLES DEL NODO MAESTRO
int nodoenvio; // Nodo al que se le envía la trama CAN
bool solicitoEnvio = false;  // Flag empleada para enviar enviar la trama CAN una sola vez
byte recTramaPS[8]; // Array donde se almacenan los byte recibidos por PS
int pos = 0; // Indice del array donde se guardan los datos del PS
byte recTramaCAN[4]; // Array donde se almacenan los datos recibidos a traves de la red CAN
String envTramaPS = ""; // Cadena que se envía a la aplicación a través del PS


//FUNCIONES DEL NODO ESCLAVO
void rxCAN_Esclavo(void);
void envia_analog(uint32_t rxnode, byte entrada);
void envia_digital(uint32_t rxnode, byte entrada);
void envia_serie(uint32_t rxnode, byte entrada);
void envia_bit(uint32_t rxnode, byte entrada);

//VARIABLES DEL NODO ESCLAVO
int dirMaestro = 0x100; // Dirección del nodo maestro

char tipoTrama; // Variable donde se almacena el tipo de trama (Analógica, digital, serie,
etc)

byte numEntrada; // Indica el numero de entrada que hay que leer
byte tramaEnvio[4]; //Trama de envío CAN

byte valorAnalogica; // Valor de la entrada analogica
byte valorDigital; // Valor de el bloque de entrada digital
byte valorSerie; // Valor recibido via serie
byte valorBit; // Valor de una entrada digital

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
```

```
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

////////////////////////////////////////////////////////////////////////////////////////
//
void setup()
{
  Serial.begin(9600); // Inicializamos puerto serie
  nodocan = IDnodo(); // Obtenemos la dirección asignada al nodo


  if (nodocan == 0x100)
    nodoMaestro = true; // La dirección del nodo maestro es la 0x100. Si se corresponde con
esa, trabajará como nodo maestro...
  else
    nodoMaestro = false; // ...sino, como modo esclavo


  initCan1(nodocan); // Inicializamos el módulo CAN 1 con la dirección de nodo indicada
  canMod1.attachInterrupt(doCan1Interrupt); // Habilitamos interrupciones del módulo CAN 1
}


void loop()
{
  //////*************FUNCIONAMIENTO COMO NODO MAESTRO*************

  if (nodoMaestro == true) // Funcionamiento del nodo como maestro
  {
      recibePS(); // Recibe la trama enviada por la aplicación a través del puerto serie
      if (solicitoEnvio == true) // Realizo lo siguiente si la trama aún no se ha enviado
      {
        // Para obtener la dirección del nodo al que se envía, se envía el valor que hay que
sumarle a 0x100 = 256 -> (nodo maestro)
        nodoenvio = 256 + recTramaPS[0];
        txCAN_Maestro(nodoenvio);  // Envio la trama CAN
        solicitoEnvio = false; // Indico que la trama ya se ha enviado
        delay(500); // Doy tiempo a recibir el mensaje
      }
      rxCAN_Maestro(); // Compruebo si he recibido algo vía CAN bus
  }


  //*************FUNCIONAMIENTO COMO NODO ESCLAVO*************
  else
  rxCAN_Esclavo(); // Compruebo si he recibido algo vía CAN bus
}



////////////////////////// DECLARACIÓN DE FUNCIONES GENERALES/////////////////

// INICIALIZACION DEL NODO CAN

void initCan1(uint32_t myaddr) {
  CAN::BIT_CONFIG canBitConfig;

  /* Se habilita el módulo CAN en modo Configuración */
  canMod1.enableModule(true);
  canMod1.setOperatingMode(CAN::CONFIGURATION);

  /* Se configura el Clock del módulo. */
  while(canMod1.getOperatingMode() != CAN::CONFIGURATION);


  canBitConfig.phaseSeg2Tq            = CAN::BIT_3TQ;
  canBitConfig.phaseSeg1Tq            = CAN::BIT_3TQ;
  canBitConfig.propagationSegTq       = CAN::BIT_3TQ;
  canBitConfig.phaseSeg2TimeSelect    = CAN::TRUE;
  canBitConfig.sample3Time            = CAN::TRUE;
  canBitConfig.syncJumpWidth          = CAN::BIT_2TQ;
  canMod1.setSpeed(&canBitConfig,SYS_FREQ,CAN_BUS_SPEED);
```

```
  /* Se asigna el área del buffer del módulo CAN */
  canMod1.assignMemoryBuffer(CAN1MessageFifoArea,2 * 8 * 16);

  /* Se configura el canal 0 para la tranmisión: buffer de 8 mensajes, RTR deshabiltado,
prioridad media/baja
   * Se configura el canal 1 para la recepción: buffer de 8 mensajes, y recepción completa del
 mensaje*/
  canMod1.configureChannelForTx(CAN::CHANNEL0,8,CAN::TX_RTR_DISABLED,CAN::LOW_MEDIUM_PRIORITY)
;
  canMod1.configureChannelForRx(CAN::CHANNEL1,8,CAN::RX_FULL_RECEIVE);

  /* Se configuran las máscaras y los filtros. */
  canMod1.configureFilter      (CAN::FILTER0, myaddr, CAN::SID);
  canMod1.configureFilterMask  (CAN::FILTER_MASK0, 0xFFF, CAN::SID,
CAN::FILTER_MASK_IDE_TYPE);
  canMod1.linkFilterToChannel  (CAN::FILTER0, CAN::FILTER_MASK0, CAN::CHANNEL1);
  canMod1.enableFilter         (CAN::FILTER0, true);

  /* Se habilitan eventos e interrupciones*/
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, true);
  canMod1.enableModuleEvent(CAN::RX_EVENT, true);


  /* Se cambia de módulo de modo Configuración a modo Normal */
  canMod1.setOperatingMode(CAN::NORMAL_OPERATION);
  while(canMod1.getOperatingMode() != CAN::NORMAL_OPERATION);
}



// CONTROL DE LAS INTERRUPCIONES CAN

void doCan1Interrupt()
{

  if ((canMod1.getModuleEvent() & CAN::RX_EVENT) != 0) {// Comprueba si la interrupción
procede la recepción de datos
    // Comprueba que evento ha generado la interrupción procede del canal 1 y analiza su
prioridad
    if(canMod1.getPendingEventCode() == CAN::CHANNEL1_EVENT) {
       /* Deshabilita la fuente del evento y sube la bandera que indica que el mensaje se ha
recibido.
        * El evento se podrá habilitar cuando se haya procesado el mensaje */
      canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, false);
      isCAN1MsgReceived = true;
    }
  }
}


// OBTENER LA DIRECCIÓN DEL NODO
int IDnodo(){
  //
 /* Se obtiene el número de nodo mediante la lecutura de un DIP Switch
  * El nodo maestro está representado por la dirección 0, y los esclavos por las demás.
  * La dirección del nodo maestro correponde a la 0x100, y la de los esclavos a partir de la 0
x128 -> 0x100 + 28h (40d)
  * Para obtener la dirección del nodo esclavo, tenemos que sumarle a la del maestro el valor
del Switch + 39 */

  int nodo;
  // Obtenemos el valor de los switch
  int offset =  digitalRead(9) + (digitalRead(10)<<1) + (digitalRead(11)<<2) +
(digitalRead(12)<<3) + (digitalRead(13)<<4);
  if (offset == 0) // Si es 0, indica que el nodo es maestro
     nodo = 0x100;
  else   // Sino, obtenemos la dirección para trabajar como esclavo
   nodo = 256 + 39 + offset;

  return nodo;
}
```

```
////////////////////////// DECLARACIÓN DE FUNCIONES DEL NODO MAESTRO/////////////////

// RECEPCION POR PUERTO SERIE DE LA TRAMA PROCEDENTE DE LA APLICACIÓN

void recibePS()
{
 if (Serial.available()) // Si está la conexión disponible...
 {
    while (Serial.available()>0) //...si hay mensaje en el buffer
    {
       delay(25);
       recTramaPS[pos] = Serial.read(); // Guardamos la informacion recibida por puerto serie
       pos++;
    }
     pos = 0;
     // Calculamos el checksum de la trama + bytes del checsum. Al ser de tipo XOR,
     // al realizar la or exlcusiva de trama+CS debemos obtener un 0.
     byte valorXOR = compruebaChecksum();

     if (valorXOR==0)
       solicitoEnvio = true; // Si la comprobación del checksum da 0, se solicita el envío de
la trama
     else
       Serial.println("1;#"); // Sino, enviamos a la aplicación la trama que indica que tiene
un error
  }
 }

// COMPROBACION DEL CHECKSUM DE LA TRAMA RECIBIDA POR PS

byte compruebaChecksum()
{
  byte acum = 0; // Acumulador del valor del checksum

  for (int i=0; i <= 3 ; i++)
     acum^=recTramaPS[i];   //Calculamos el checksum acumulado de la trama recibida por PS
  return acum;
}

// TRANMISIÓN A TRAVES DE LA RED CAN

void
txCAN_Maestro(uint32_t rxnode) {

  CAN::TxMessageBuffer * message;

  message = canMod1.getTxMessageBuffer(CAN::CHANNEL0);

  if (message != NULL) {
    // Limpiamos el buffer
    message->messageWord[0] = 0;
    message->messageWord[1] = 0;
    message->messageWord[2] = 0;
    message->messageWord[3] = 0;

    message->msgSID.SID   = rxnode;   //Nodo de destino
    message->msgEID.IDE   = 0;
    message->msgEID.DLC   = 2;        // Indicamos el numero de bytes que se envían en el
campo de datos de la trama

    // Introducimos los valores en el campo de datos
    message->data[0]      = recTramaPS[1];
    message->data[1]      = recTramaPS[2];

    // message->data[2]        =
    // message->data[3]        =
    // message->data[4]        =
    // message->data[5]        =
    // message->data[6]        =
    // message->data[7]        =

    // Esta función permite al módulo CAAN saber que el mensaje está listo para ser procesado
     canMod1.updateChannel(CAN::CHANNEL0);
```

```
    // Se envía el mensaje a través del canal TX
     canMod1.flushTxChannel(CAN::CHANNEL0);
  }
}

// RECEPCIÓN VIA CAN
void rxCAN_Maestro(void) {

  CAN::RxMessageBuffer * message;

  if (isCAN1MsgReceived == false) {
  /* Si la bandera que indica que se ha recibido un mensaje está bajada
   * se sale de la función*/
    return;
  }

  /* El mensaje se ha recibido. Se baja la bandera para coger el mensaje*/
  isCAN1MsgReceived = false;

  message = canMod1.getRxMessage(CAN::CHANNEL1);


  for (int i=0;i<=3;i++)
  {
  recTramaCAN[i] = message->data[i]; //Guardamos cada byte del campo de datos de la trama
recibida

  // Se transforma la trama CAN en una trama con formato "
dato[0];dato[1];dato[2];dato[3];CS;# " para enviarla al PC
  envTramaPS+=(String)recTramaCAN[i];
  envTramaPS+=";";
  }

   envTramaPS+=(String)calculaChecksum();
   envTramaPS+=";#";


  Serial.println(envTramaPS); // Se envía la trama ya estructurada por puerto serie al PC
  envTramaPS="";

  /* Esta función permite saber al módulo CAN que el mensaje se ha procesado.
   * Se habilita el evento mediante el módulo pueda generar una interrupción
   * cuando se reciba un mensaje*/
  canMod1.updateChannel(CAN::CHANNEL1);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, true);

}

// FUNCIÓN QUE CALCULA EL CHECKSUM DE UNA TRAMA

byte calculaChecksum()
{
  byte acum = 0; // Acumulador del valor del checksum
  for (int i=0; i <= 3 ; i++)
     acum^=recTramaCAN[i];    // Calculamos el checksum de la entrada recibida por CAN, que es
la que enviaremos por PS

  acum^= ';'; // El checksum también tiene que incluir el los caracteres de fin de trama
  acum^= '#';
  return acum;
}



//////// DECLARACIÓN DE FUNCIONES DEL NODO ESCLAVO ////////////////////////////

// RECEPCIÓN VIA CAN
void rxCAN_Esclavo(void) {

  CAN::RxMessageBuffer * message;

  if (isCAN1MsgReceived == false) {
```

```
      /* Si la bandera que indica que se ha recibido un mensaje está bajada
   * se sale de la función*/
    return;
  }


   /* El mensaje se ha recibido. Se baja la bandera para coger el mensaje*/
  isCAN1MsgReceived = false;

  message = canMod1.getRxMessage(CAN::CHANNEL1);

   // Se recibe el byte 0 del campo de datos
   // Identificamos el tipo de entrada de los datos solicitados
   // -> A = analógico //  D = digital (byte) // S = serie // B = ditial (bit)
   tipoTrama = message->data[0];

  if (tipoTrama=='A'){   // Datos sobre entrada Analógica

    numEntrada=message->data[1]; // El numero de entrada se indica en el byte 1 del campo de
datos
    envia_analog(dirMaestro,numEntrada);// Se solicita una lectura de entrada analógica
      }

  else if (tipoTrama=='D'){   // Datos sobre entrada de datos en forma de byte

    numEntrada=message->data[1]; // El numero de entrada se indica en el byte 1 del campo de
datos
   envia_digital(dirMaestro,numEntrada); // Se solicita una lectura de entrada digital
  }

  else if (tipoTrama=='S'){   // Datos sobre entrada serie

    numEntrada=message->data[1];  // El numero de entrada del par TX,RX se indica en el byte 1
del campo de datos
    envia_serie(dirMaestro,numEntrada); // Se solicita una lectura de entrada serie
  }

   else if (tipoTrama=='B'){   // Datos sobre el estado una entrada bit

    numEntrada=message->data[1];  // El numero de entrada se indica en el byte 1 del campo de
datos
    envia_bit(dirMaestro,numEntrada); // Se solicita una lectura de entrada bit
  }

  /* Esta función permite saber al módulo CAN que el mensaje se ha procesado.
   * Se habilita el evento mediante el módulo pueda generar una interrupción
   * cuando se reciba un mensaje*/

  canMod1.updateChannel(CAN::CHANNEL1);
  canMod1.enableChannelEvent(CAN::CHANNEL1, CAN::RX_CHANNEL_NOT_EMPTY, true);

}


// ENVIO CAN DE ENTRADA ANALOGICA

void envia_analog(uint32_t rxnode, byte entrada){

  int inAnalogica = A0; // Pin de la entrada analógica
  switch (entrada) { // En función del numero de entrada que se haya obtenido, se habilita esa
entrada

    case '0': inAnalogica=A0; break;
    case '1': inAnalogica=A1; break;
    case '2': inAnalogica=A2; break;
    case '3': inAnalogica=A3; break;
    case '4': inAnalogica=A4; break;
    case '5': inAnalogica=A5; break;
    case '6': inAnalogica=A6; break;
    case '7': inAnalogica=A7; break;
    case '8': inAnalogica=A8; break;
    case '9': inAnalogica=A9; break;
    case 'A': inAnalogica=A10; break;
```

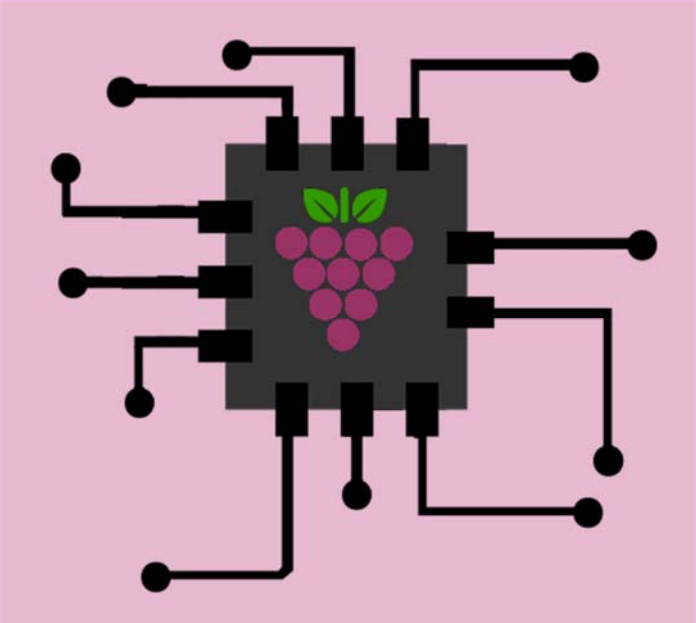
```
    case 'B': inAnalogica=A11; break;
    case 'C': inAnalogica=A12; break;
    case 'D': inAnalogica=A13; break;
    case 'E': inAnalogica=A14; break;
    case 'F': inAnalogica=A15; break;
    default: break;//ERROR ANALOGICA
  }
  int lecturaAnalogica = analogRead(inAnalogica); //Se lee esa entrada
  valorAnalogica = map(lecturaAnalogica, 0, 1023, 0, 255);   // La escalamos a 0-255 para
poder enviar el dato en 1 byte
  //lcd.clear();
  //lcd.print("ANALOGICA ");
//lcd.print(char(entrada));
 // lcd.print("=");
  //lcd.print(String(valorAnalogica));
  //Serial.println("ANALOGICA");
  delay(500);
  // Guardamos los datos en un array que introduciremos en el campo de datos de la trama que
se enviará al nodo maestro
  tramaEnvio[0] = nodocan;
  tramaEnvio[1] = tipoTrama;
  tramaEnvio[2] = entrada;
  tramaEnvio[3] = valorAnalogica;

  txCAN1(dirMaestro,tramaEnvio); // Se envía la trama
}

// ENVIO CAN DE ENTRADA DIGITAL

void envia_digital(uint32_t rxnode, byte entrada){
  int byteDigital[8]; // Pines del bloque de la entrada digital (byte)

  switch (entrada) {// En función del numero de entrada que se haya obtenido, se habilita esa
entrada

  case '0': byteDigital={38,40,42,44,46,48,50,52}; break;
  case '1':  byteDigital={39,41,43,45,47,49,52,53}; break;
  default :break; // ERROR ENTRADA DIGITAL BYTE
  }
  for(int i=0;i<=7;i++)
    pinMode(byteDigital[i],INPUT);

  valorDigital= digitalRead(byteDigital[0]) + (digitalRead(byteDigital[1])<<1) +
(digitalRead(byteDigital[2])<<2)
  + (digitalRead(byteDigital[3])<<3) + (digitalRead(byteDigital[4])<<4) +
(digitalRead(byteDigital[5])<<5)
  + (digitalRead(byteDigital[6])<<6) + (digitalRead(byteDigital[7])<<7); //Obtenemos un valor
0-255 de las 8 entradas digitales

//lcd.clear();
//lcd.print("BLOQ BYTE ");
//lcd.print(char(entrada));
//lcd.print("=");
//lcd.print(String(valorDigital));

//Serial.println("BLOQ BYTE");
  delay(500);
  // Guardamos los datos en un array que introduciremos en el campo de datos de la trama que
se enviará al nodo maestro
  tramaEnvio[0] = nodocan;
  tramaEnvio[1] = tipoTrama;
  tramaEnvio[2] = entrada;
  tramaEnvio[3] = valorDigital;

  txCAN1(dirMaestro,tramaEnvio);// Se envía la trama
}

// ENVI0 CAN DE ENTRADA SERIE


void envia_serie(uint32_t rxnode, byte entrada){
```

```
  switch (entrada){  // En función de la entrada se habilita el canal serie correspondiente y
se lee el valor
  case 1:
    Serial1.begin(9600);
    if (Serial1.available() > 0) {
      valorSerie = Serial1.read();}
     Serial1.end();
   break;

   case 2:
   Serial2.begin(9600);
    if (Serial2.available() > 0) {
      valorSerie = Serial2.read();}
     Serial2.end();

   case 3:
   Serial3.begin(9600);
    if (Serial3.available() > 0) {
      valorSerie = Serial3.read();}
     Serial3.end();
   break;

   default:break; }//ERROR SERIE

 //lcd.clear();
 //lcd.println("SERIE");
 //Serial.println("SERIE");
  delay(500);

  // Guardamos los datos en un array que introduciremos en el campo de datos de la trama que
se enviará al nodo maestro
   tramaEnvio[0] = nodocan;
  tramaEnvio[1] = tipoTrama;
  tramaEnvio[2] = entrada;
  tramaEnvio[3] = valorSerie;
  txCAN1(dirMaestro,tramaEnvio);

}

// ENVIO CAN DE ENTRADA BIT

void envia_bit(uint32_t rxnode, byte entrada){
 int pin = entrada;
  pinMode(pin,INPUT);
 valorBit = digitalRead(pin); // Pines del bloque de la entrada digital (byte)

//lcd.clear();
//lcd.print("BIT ");
//lcd.print(char(pin));
//lcd.print(" = ");
//lcd.print(String(valorBit));


//Serial.println("BIT");
  delay(500);

  tramaEnvio[0] = nodocan;
  tramaEnvio[1] = tipoTrama;
  tramaEnvio[2] = entrada;
  tramaEnvio[3] = valorBit;

  // Se envía la trama
  txCAN1(dirMaestro,tramaEnvio);

}

// ENVIO DE TRAMA CAN

void txCAN1(uint32_t rxnode, byte trama[4]) {

  CAN::TxMessageBuffer * message;

  message = canMod1.getTxMessageBuffer(CAN::CHANNEL0);
```

```
  if (message != NULL) {
    // clear buffer
    message->messageWord[0] = 0;
    message->messageWord[1] = 0;
    message->messageWord[2] = 0;
    message->messageWord[3] = 0;

    message->msgSID.SID   = rxnode;   //Nodo de destino
    message->msgEID.IDE   = 0;
    message->msgEID.DLC   = 4;                  // Indicamos el numero de bytes que tiene el
campo de datos de la trama que se envia

   // Se introducen los datos en el campo de datos
    message->data[0]      = trama[0];
    message->data[1]        = trama[1];
    message->data[2]        = trama[2];
    message->data[3]        = trama[3];
    message->data[4]        = 0;
    message->data[5]        = 0;
    message->data[6]        = 0;
    message->data[7]        = 0;


          for (int i=0; i <= 3 ; i++)
          Serial.print(trama[i]);
          // Se envía la trama por puerto serie para poder hacer comprobaciones

          Serial.println();
  // Esta función permite al módulo CAN saber que el mensaje está listo para ser procesado
   canMod1.updateChannel(CAN::CHANNEL0);
  // Se envía el mensaje a través del canal TX
    canMod1.flushTxChannel(CAN::CHANNEL0);
  }
}
```

MANUAL DE USUARIO

EnoCAN



Daniel Pérez García

UNIVERSIDAD DE LA RIOJA

CONTENIDO

ILUSTRACIONES

1. INSTALACIÓN DE MYSQL WORKBENCH 8.0

MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL.

A través de esta herramienta se va a configurar la ubicación donde se va a albergar la base de datos, así como la gestión de usuarios para su acceso. Funciona bajo licencia pública, y se puede descargar desde su página web:

https://dev.mysql.com/downloads/workbench/

2. CREAR NUEVA CONEXIÓN

Para poder acceder a la base de datos desde la aplicación lo primero que hay que hacer es crear una nueva conexión. Esta conexión puede ser de tipo local (LocalHost) o a un servidor externo.

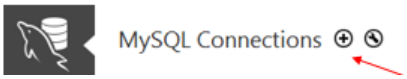


Ilustración 1- Crear nueva conexión MySQL

Al configurar la nueva conexión, hay que establecer que se realice a través del método TCP/IP. Para una conexión local, hay que asignar el servidor LocalHost que hace referencia al propio equipo, o la IP equivalente 127.00.1. Si se desea configurar un servidor remoto, hay que introducir la IP del servidor. Tanto en servidor local como en remoto, el puerto estándar para la conexión es el 3306. Por último, se debe asignar una contraseña al usuario ‘root’ del sistema, el cual tendrá todos los permisos.

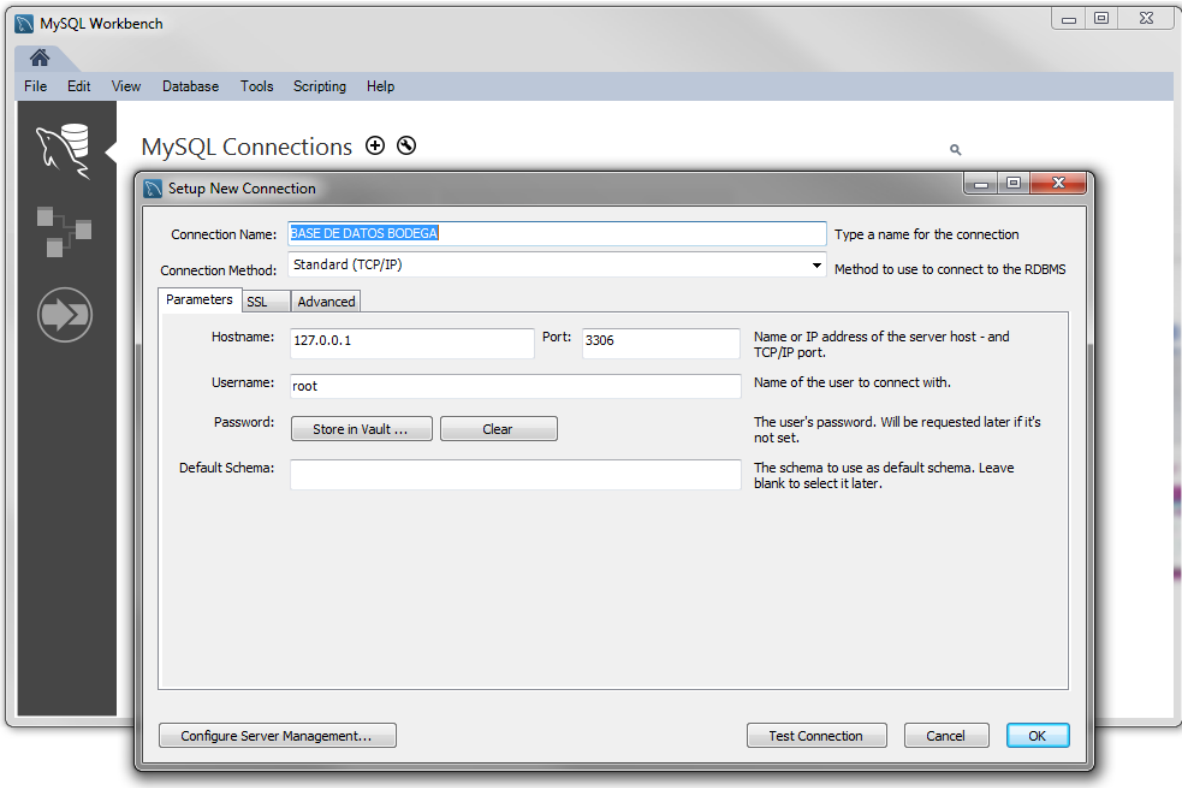


Ilustración 2 - Configurar nueva conexión

Una vez creada la conexión ya se podrá acceder a ella tanto desde la aplicación *EnoCAN* como desde MySQL Workbench.

3. ADMINISTRACIÓN DE USUARIOS

Una vez creada la conexión, únicamente se puede acceder a ella desde el usuario ‘root’, el cual posee todos los permisos. Para crear nuevos usuarios que puedan acceder a la base de datos, cada uno con sus permisos correspondientes, se debe acceder a la sección de ‘Usuarios y Privilegios’.

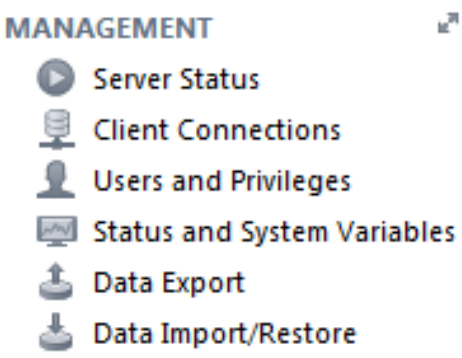


Ilustración 3 - Adeministración de usuarios

Desde esta sección se pueden configurar los permisos de los usuarios ya existentes así como crear nuevos usuarios.

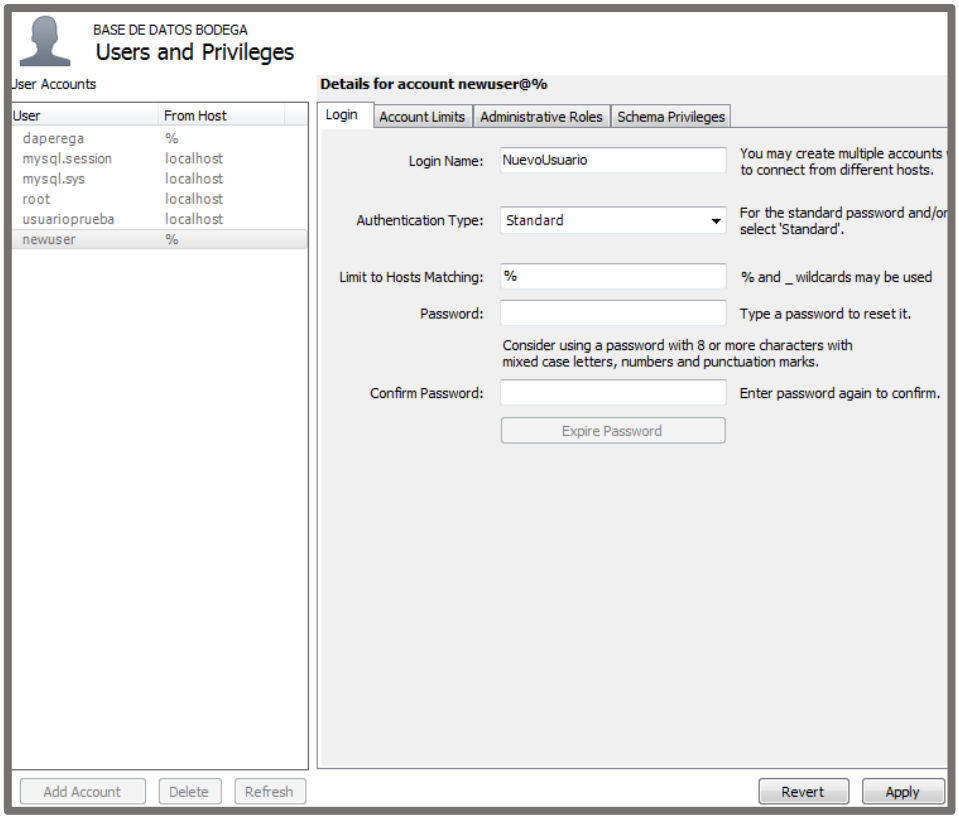


Ilustración 4 - Ventana de administración de usuarios

4. CONFIGURACIÓN DEL SISTEMA

Una vez creados los usuarios de acceso a la base de datos, se puede arrancar la aplicación EnoCAN. Aparecerá la siguiente ventana:

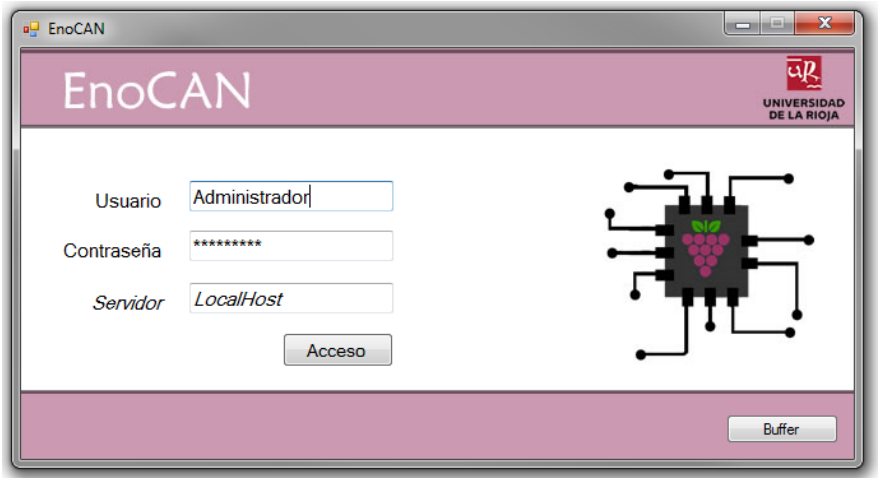


Ilustración 5- Ventana de Inicio de la aplicación

En ella aparecen tres campos a rellenar: un usuario, contraseña y el servidor. En ellos se debe introducir una cuenta de usuario válida registrada en la base de datos, como se ha explicado en el apartado anterior. En el campo servidor habrá que introducir LocalHost (o bien 127.0.0.1) si se trata de un servidor local, o si se trata de servidor remoto su IP correspondiente. Si no se consigue acceder a la base de datos bien por no disponer de una cuenta o por un error en la conexión, el sistema puede trabajar en el denominado 'Modo Buffer'. Los modos de operación se explicarán en el siguiente apartado.

Tanto si accedemos en "Modo Normal" como en "Modo Buffer", al acceder al sistema lo primero que se tendrá que seleccionar es el número de nodos. Posteriormente en la ventana de configuración vendrá representado el modo de operación en una etiqueta. Se podrán configurar hasta 10 nodos.

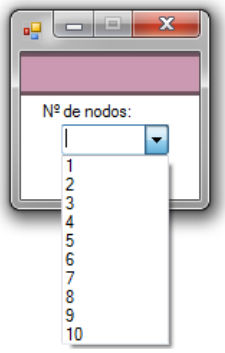


Ilustración 6 - Selección del número de nodos

Tras seleccionar el número de nodos del sistema, aparecerá la pantalla principal para su configuración. A continuación, se van a explicar los diferentes campos que la forman.

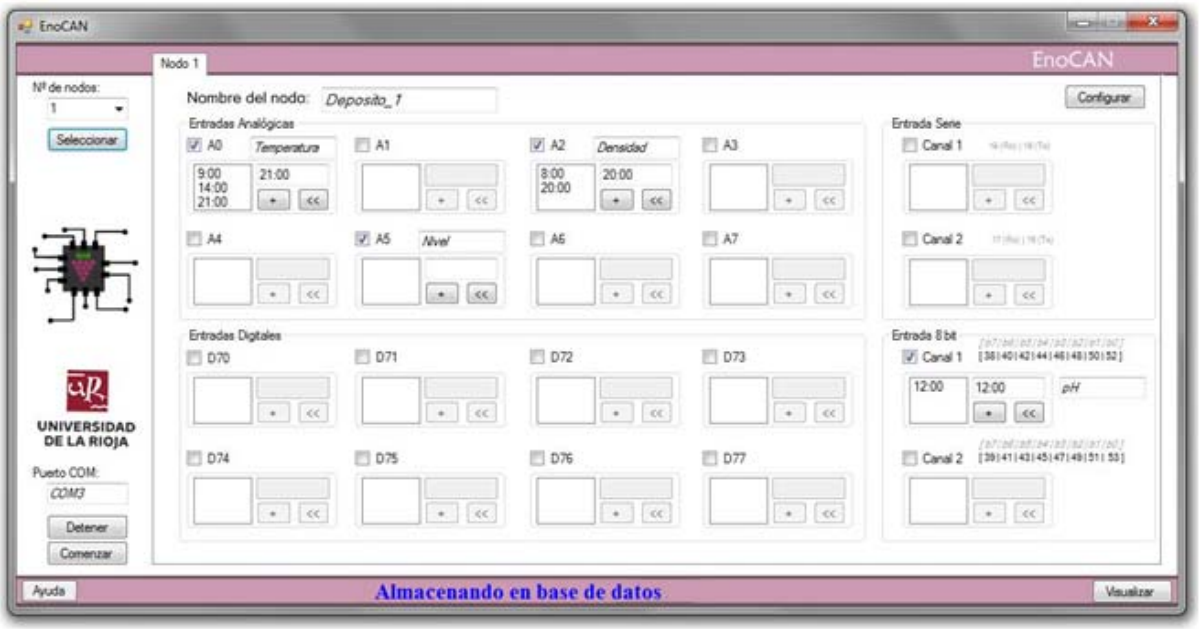


Ilustración 7 - Ventana de configuración de la aplicación

- ✓ En función del nº de nodos seleccionados, aparecerá en la zona superior una lista de pestañas correspondientes a cada nodo. Dentro de cada pestaña podremos configurar el funcionamiento de cada nodo.

- ✓ En el campo “nombre del nodo” corresponde al nombre que se desea dar al esquema de ese nodo.

- ✓ Para cada nodo hay disponibles 8 entradas digitales, 8 entradas analógicas, 2 entradas de byte y 2 canales para la comunicación serie. En cada entrada viene indicado el pin al que se corresponde. Cuando se selecciona cada entrada, se desbloquea la zona donde se introducen las horas a las que se desea que se tome una medida. Además, aparece junto a la entrada un campo donde se puede introducir el nombre que se le desea dar a la tabla correspondiente a esa entrada en la base de datos.

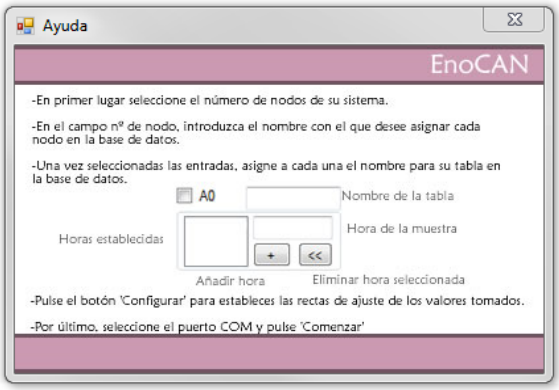


Ilustración 8 - Ventana de ayuda

Para cada muestra, se recibe desde la red de nodos un valor comprendido entre 0 y 255. Pulsando el botón "Configurar", aparecerá una ventana que permitirá establecer las ecuaciones características de los sensores empleados para adecuar el valor recibido de la red de sensores a valores en unidades reales.

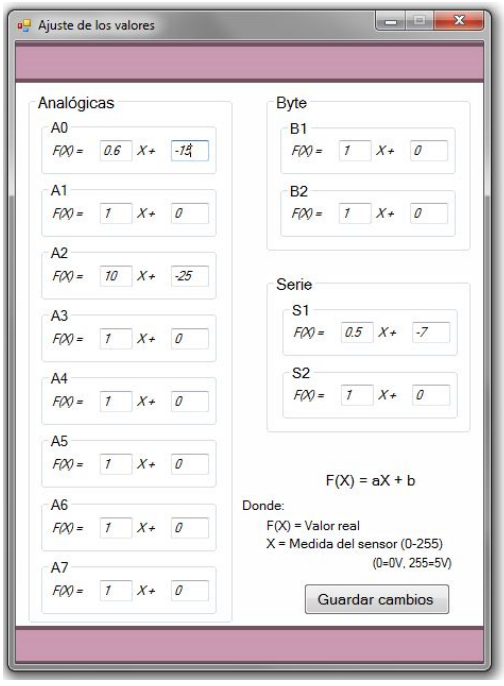


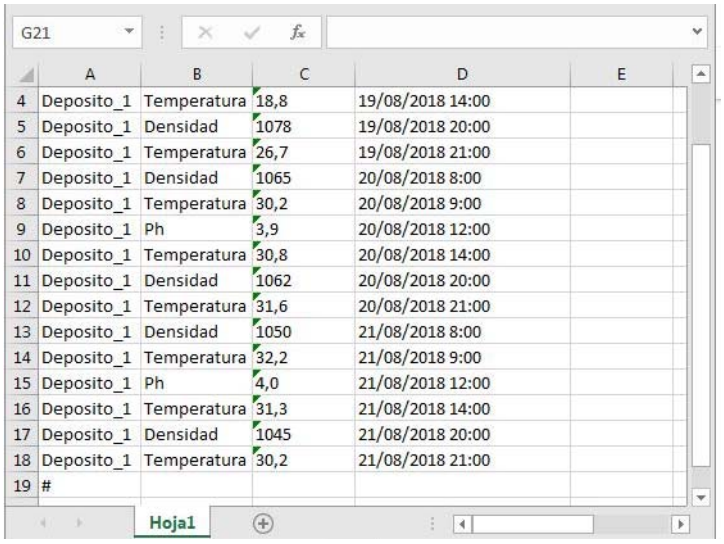
Ilustración 9 - Configuración de las funciones de transferencia

Una vez configurado el funcionamiento de la red de nodos, se puede proceder a conectar el sistema. Para ello se debe establecer en el campo "Puerto COM" el puerto a través del cual se conectará la placa correspondiente al nodo maestro con el PC a través de USB 2.0.

Con todo realizado, ya se podrá proceder a la puesta en marcha del sistema pulsando el botón “Comenzar”. Tras esto aparecerá la etiqueta “Sistema conectado” y no se permitirá cambiar la configuración mientras esté conectado. Para poder cambiar la configuración, habrá que pulsar el botón “Detener” y se desbloquearan los campos que permitan modificar el sistema.

5. MODOS DE TRABAJO

Cuando la aplicación recibe los valores de los parámetros, éstos no se guardan directamente en la base de datos, sino que se introducen en un buffer. Se trata de un archivo Excel ubicado en el PC que la aplicación crea automáticamente. Funciona como una memoria FIFO, en la cual el primer dato que entra es el primero que sale



	A	B	C	D	E
4	Deposito_1	Temperatura	18,8	19/08/2018 14:00	
5	Deposito_1	Densidad	1078	19/08/2018 20:00	
6	Deposito_1	Temperatura	26,7	19/08/2018 21:00	
7	Deposito_1	Densidad	1065	20/08/2018 8:00	
8	Deposito_1	Temperatura	30,2	20/08/2018 9:00	
9	Deposito_1	Ph	3,9	20/08/2018 12:00	
10	Deposito_1	Temperatura	30,8	20/08/2018 14:00	
11	Deposito_1	Densidad	1062	20/08/2018 20:00	
12	Deposito_1	Temperatura	31,6	20/08/2018 21:00	
13	Deposito_1	Densidad	1050	21/08/2018 8:00	
14	Deposito_1	Temperatura	32,2	21/08/2018 9:00	
15	Deposito_1	Ph	4,0	21/08/2018 12:00	
16	Deposito_1	Temperatura	31,3	21/08/2018 14:00	
17	Deposito_1	Densidad	1045	21/08/2018 20:00	
18	Deposito_1	Temperatura	30,2	21/08/2018 21:00	
19	#				

Ilustración 10 - Contenido del buffer tras días de funcionamiento

Cuando el sistema funciona en “Modo Normal”, la aplicación comprueba continuamente si hay datos dentro del buffer, y si los hay, los introduce dentro de la base de datos y los elimina del buffer. A parte del valor y la fecha de la muestra, también se guarda el nodo y la entrada a la que pertenece. De esta manera la aplicación sabe en qué base de datos y tabla debe introducir la muestra. Con el sistema funcionando en “Modo Buffer” los datos se van almacenando en este archivo sin ser enviados a la base de datos. En el momento que se retome la conexión con el servidor, todos los datos almacenados se volcarán en la base de datos.

Si mientras el sistema está funcionando se produce algún error en la conexión con la base de datos, el sistema automáticamente cambia a “Modo Buffer”.

También se puede acceder a este modo pulsando el botón “Buffer” de la pantalla de inicio. Esto es útil si por ejemplo no se dispone de un usuario y contraseña de acceso a la base de datos. Para ello, tras arrancar la aplicación, se introduce el usuario y contraseña asignados a este modo de operación

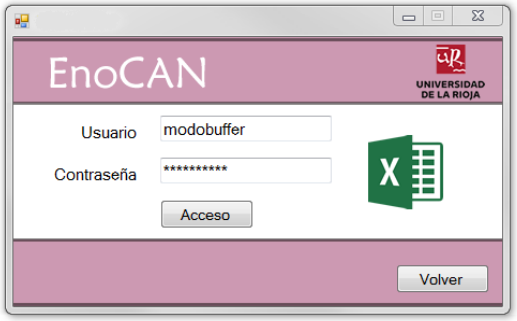


Ilustración 11 - Ventana de acceso en Modo Buffer

- Usuario: *modobufer*
- Contraseña: *passmodobufer*

De esta manera se accederá a la ventana de configuración, pero en la zona inferior se indicará que se está trabajando en este modo.

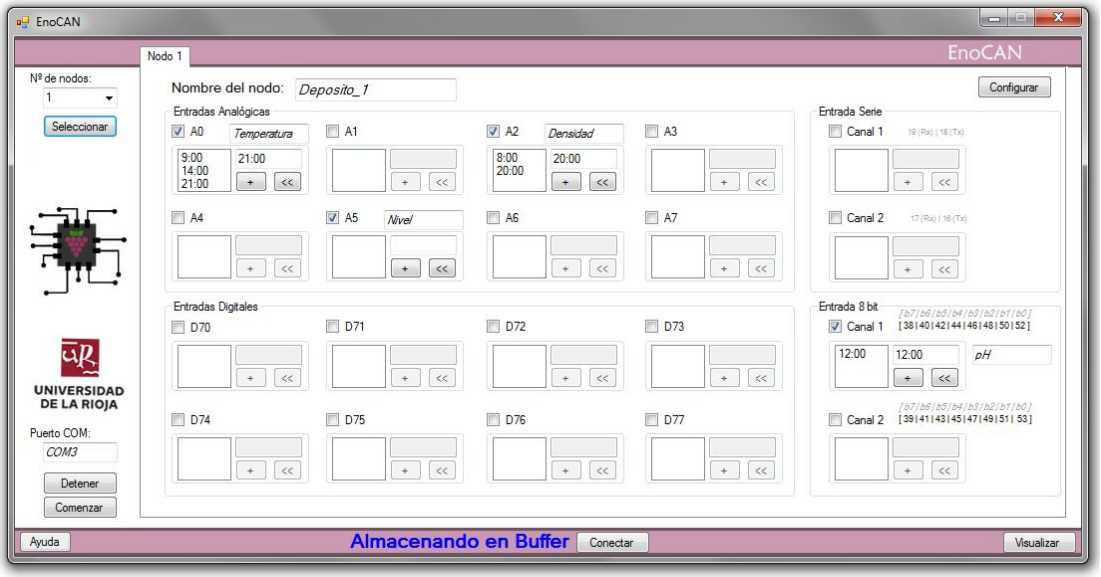


Ilustración 12 - Sistema en Modo Buffer

Para volver a conectar la aplicación basta con pulsar el botón “Conectar” de la parte inferior central. Aparecerá la ventana de inicio en la que se debe introducir el usuario, contraseña y servidor donde se aloja la base de datos. Si la conexión es correcta el sistema volverá a trabajar en “Modo Normal”.

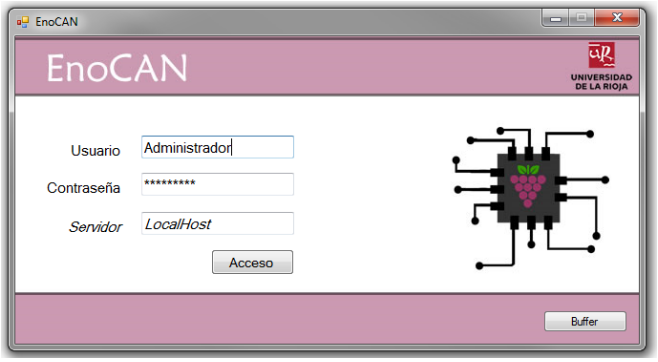


Ilustración 13 - Acceso al sistema en Modo Normal

6. REPRESENTACIÓN DE DATOS

Al pulsar el botón "Visualizar" situado en la parte inferior derecha de la ventana de configuración, se podrá acceder a la ventana desde la que se pueden representar los datos de la base.

Desde esta ventana se puede acceder a los datos almacenados en la base de datos para su representación bien en forma de tabla o de gráfico. Para acceder a ellos se debe introducir el nombre del nodo (base) que se quiere consultar y la entrada (tabla) perteneciente a ese nodo. Una vez representados los datos en forma de tabla, se puede proceder a su representación gráfica. Para ello se debe seleccionar qué columna de la tabla se representa en cada eje.

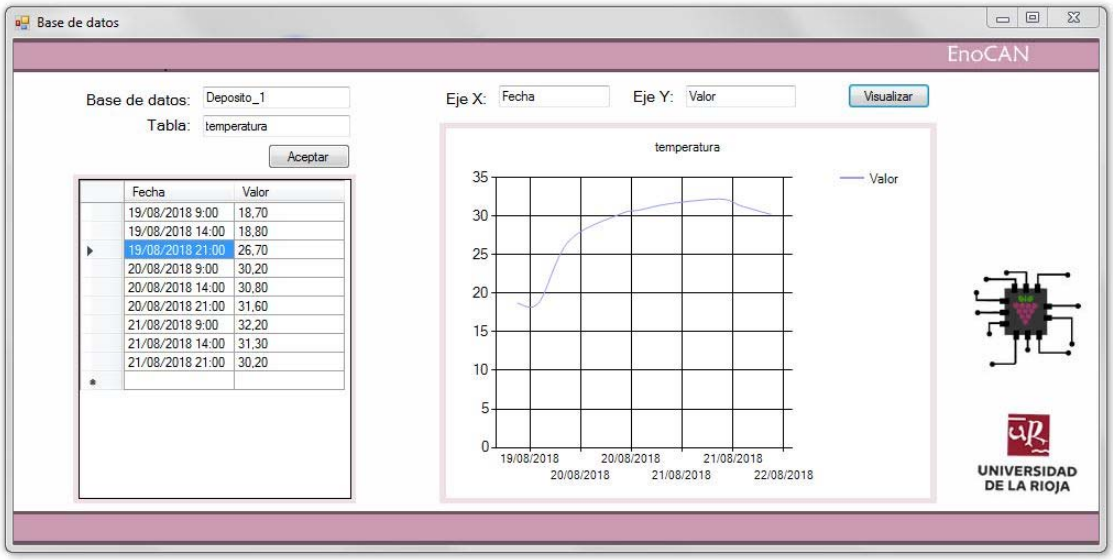


Ilustración 14 - Representación de la temperatura

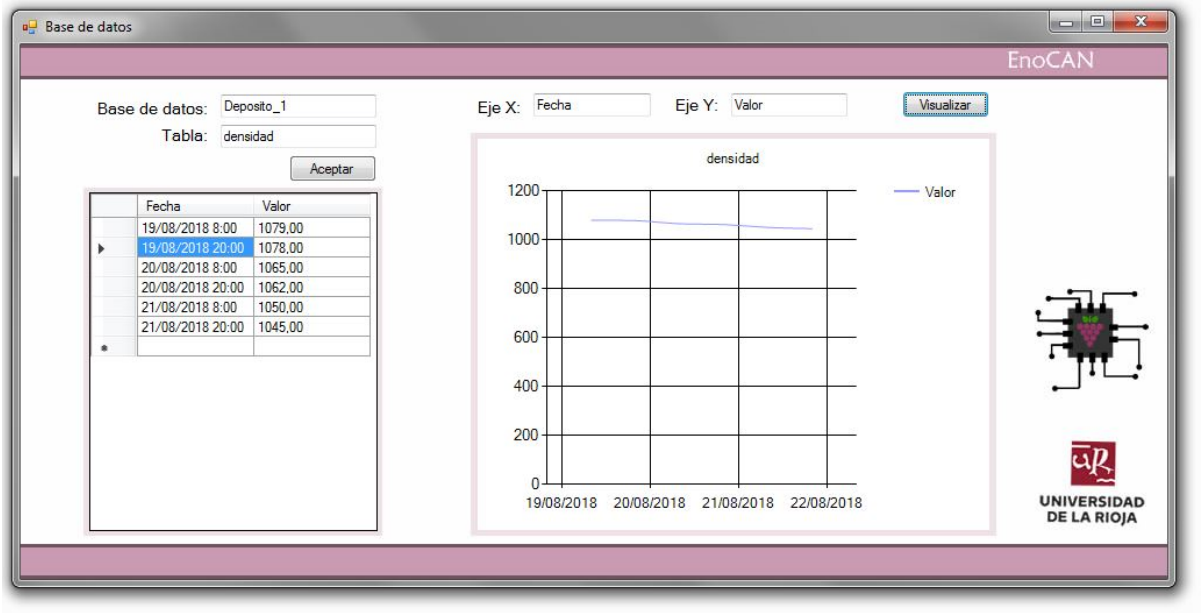


Ilustración 15 - Representación de la densidad

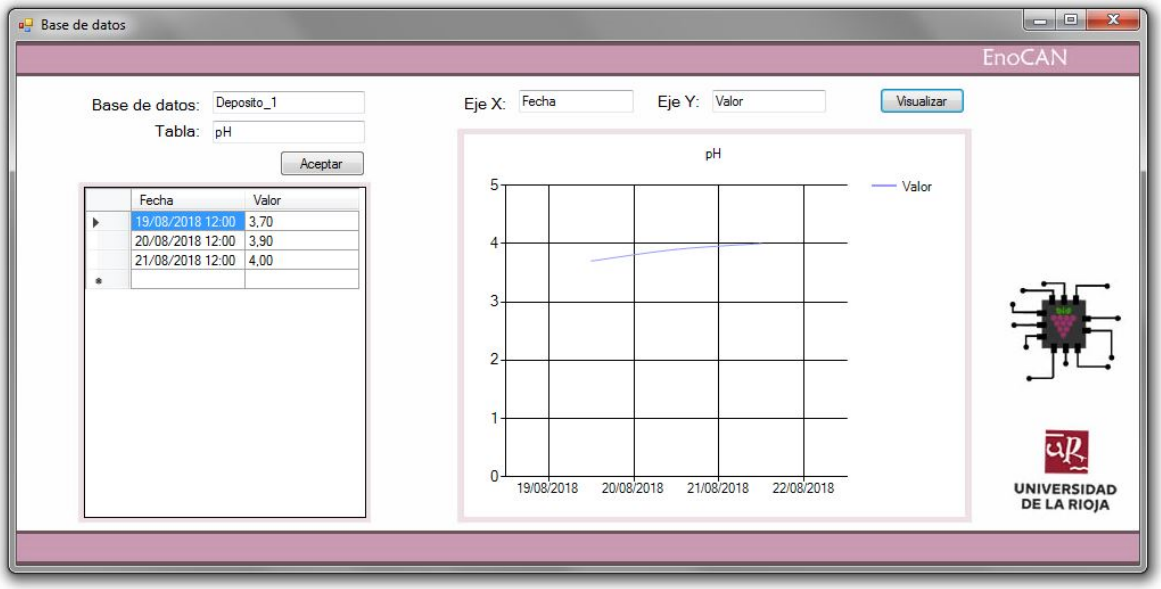


Ilustración 16 - Representación del Ph

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

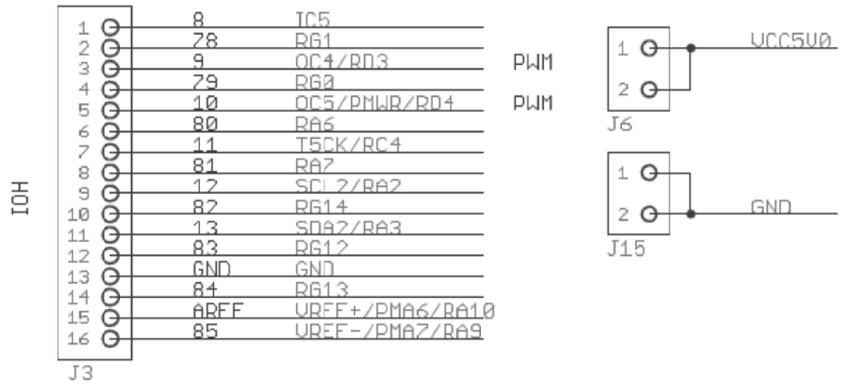
Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

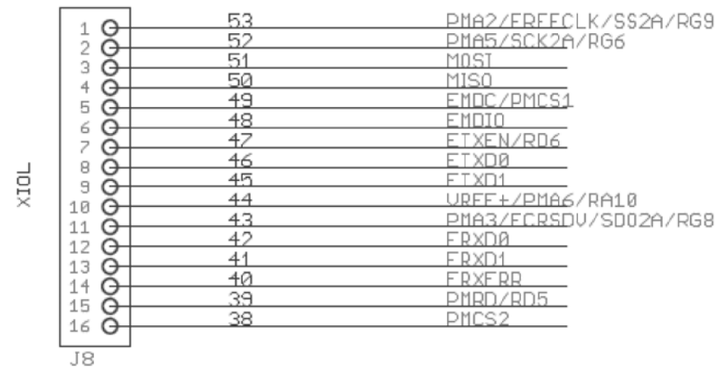
4. PLANOS

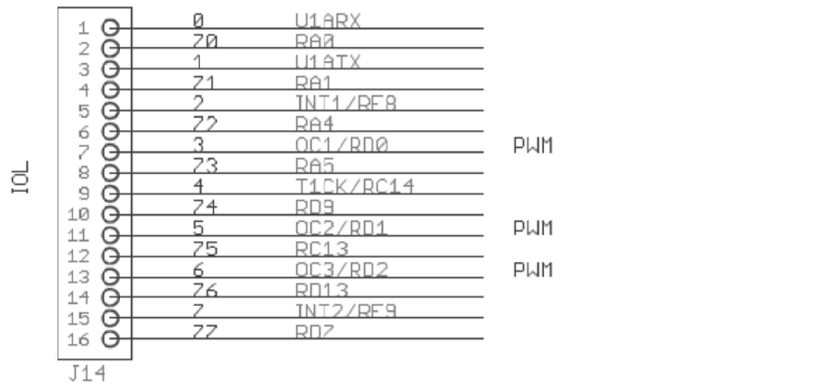
Daniel Pérez García

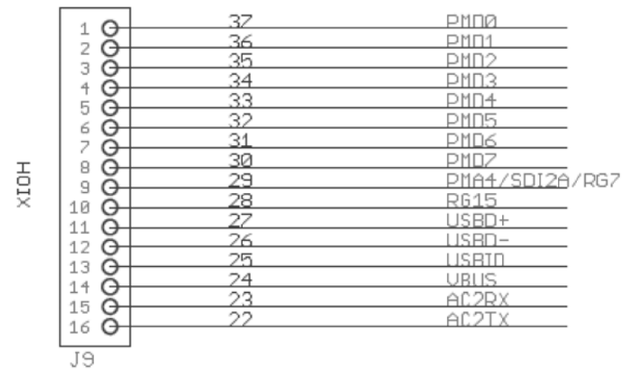
Septiembre 2018

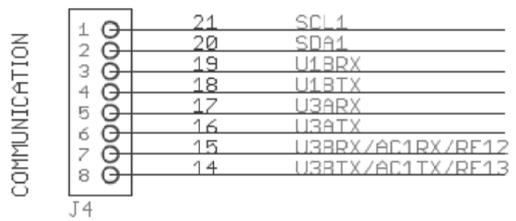
ÍNDICE DE PLANOS

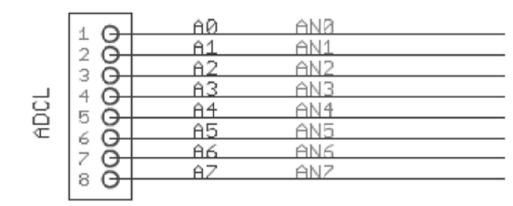


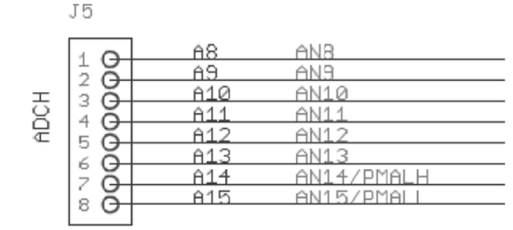


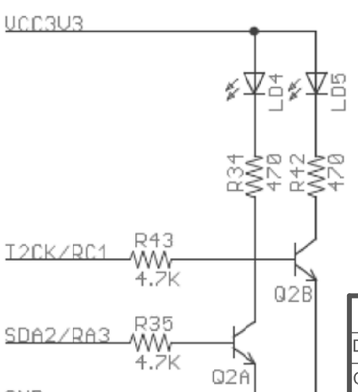


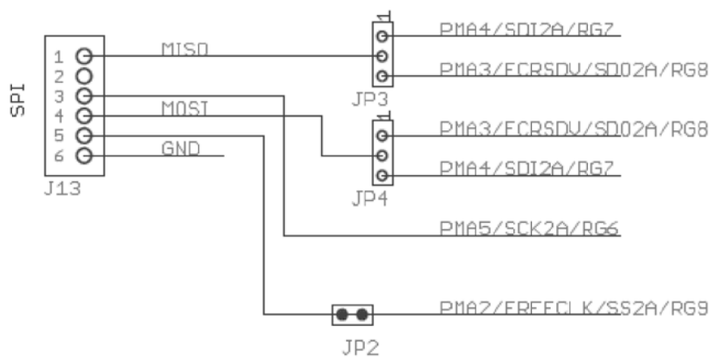




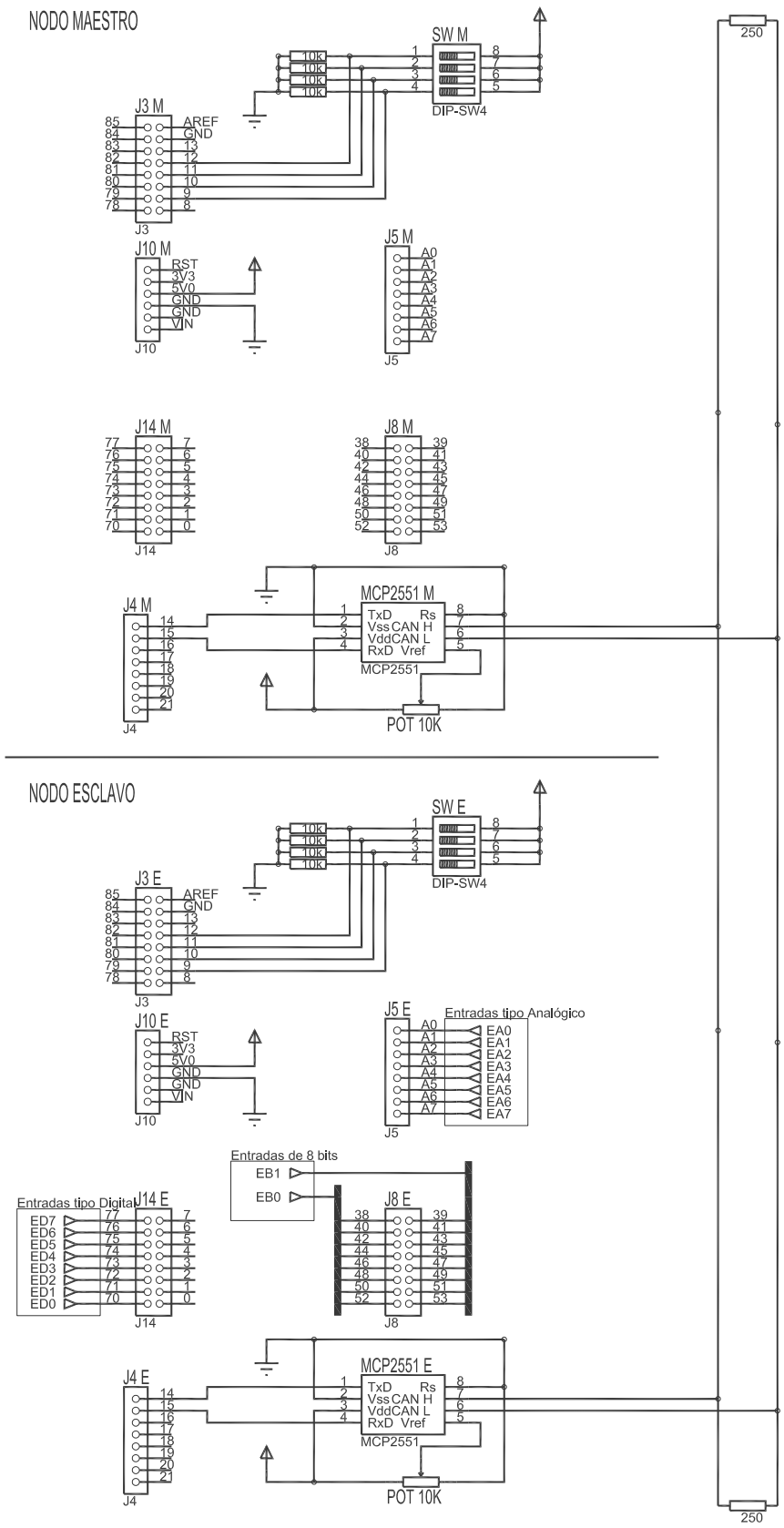








	Fecha	Nombre		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en Ingeniería Electrónica Industrial y Automática	
Dibujado	25/07/2018	Daniel Pérez			
Comprobado					
Id.s.normas	U.N.E	Tolerancia general			
Escala: SE	Conectores E/S de la placa ChipKIT MAX 32			Nº de plano: 1	
				Sustituye a	
				Sustituido por	



	Fecha	Nombre	
Dibujado	25/07/2018	Daniel Pérez	
Comprobado			
Id.s.normas	U.N.E	Tolerancia general	

ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INDUSTRIAL
UNIVERSIDAD DE LA RIOJA
Grado en Ingeniería Electrónica Industrial y Automática

Escala	Esquema de montaje de nodo maestro y nodo esclavo	Nº de plano
SE		2
		Sustituye a
		Sustituido por

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

5. PLIEGO DE CONDICIONES

Daniel Pérez García

Septiembre 2018

ÍNDICE DEL PLIEGO DE CONDICIONES

5.1 INTRODUCCIÓN

5.1.1 Objeto

El autor del presente proyecto ha cursado los estudios de Grado en Ingeniería Electrónica Industrial y Automática, cumpliendo con la normativa establecida por la Escuela Superior de Ingeniería Industrial, bajo la designación de "Trabajo Fin de Grado".

El objeto de este Pliego de Condiciones es exponer todas las disposiciones y condiciones técnicas, administrativas y económicas, así como las especificaciones y normativa que se debe cumplir a la hora de llevar a cabo la realización del proyecto y su implantación en un proceso real, aunque quede fuera del alcance del presente proyecto.

Este proyecto se trata de un prototipo, por lo que las especificaciones expuestas serán consideradas en el caso de implantar físicamente el prototipo en procesos reales.

En caso de no realizarse estas condiciones, el proyectista no se responsabilizará de posibles fallos o averías que se puedan ocasionar durante el funcionamiento, repercutiendo los problemas derivados sobre terceras personas.

Toda modificación del proyecto deberá ser aprobada por el ingeniero o proyectista.

5.1.2 Propiedad intelectual

La propiedad intelectual del autor y director del "Trabajo Fin de Grado" se regirá según el artículo 12 del reglamento de "Proyectos Fin de Carrera" de la "Escuela Técnica Superior de Ingeniería Industrial de La Rioja", aprobado por el Consejo de Gobierno el 15 de abril de 2005.

5.2 CONDICIONES GENERALES

5.2.1 Condiciones Administrativas

El proyecto consta de los siguientes documentos:

- Un Índice General con los índices de cada documento básico que forma el proyecto.
- Una Memoria donde se consideran las necesidades a satisfacer y los factores técnicos a tener en cuenta profundizando en las posibles soluciones técnicas y la justificación de la solución elegida.
- Unos Anexos donde se recoge la documentación considerada de interés para ampliar la descripción tanto del protocolo como de los componentes empleados, como sus hojas de características. El código de la programación realizada tanto para el firmware de las placas como de la aplicación.
- Planos de interconexión de los elementos para la conexión de los nodos del sistema a la red.
- Un Pliego de Condiciones donde se describe las condiciones técnicas, administrativas y económicas para la materialización del proyecto y así evitar malas interpretaciones.
- Un Presupuesto donde se recoge el coste de los componentes utilizados y la suma total que junto a la mano de obra se obtenga el coste final de una supuesta implementación real.

5.2.2 Normativa y Reglamentación

Este proyecto está sujeto tanto a la normativa española como a la normativa internacional. Debido a sus características se rige bajo el Reglamento Electrotécnico de Baja Tensión. Por otro lado, en AENOR se pueden encontrar las siguientes normativas:

- UNE 21302 – Vocabulario electrotécnico.
- UNE EN61000 – Compatibilidad electromagnética.
- UNE 20 050 74 – Código para marcado de resistencias y condensadores.
- UNE 20 531 73 – Código de colores para resistencias y condensadores.

La especificación para el protocolo CAN para la comunicación de la red de nodos, está publicada en el estándar ISO 11898.

En el supuesto de que se fuera a comercializar en la Unión Europea, para la obtención del marcado CE deberá someterse a los ensayos de compatibilidad electromagnética y cualquiera de las preceptivas condiciones de ensayo que defina la normativa de comercialización.

5.3 CONDICIONES DE MATERIALES Y EQUIPOS

Todos los componentes y materiales utilizados en este proyecto, deben cumplir las especificaciones técnicas descritas tanto en los Anexos como en el siguiente apartado.

5.3.1 Componentes electrónicos

ChipKit MAX32

Las especificaciones y características de la placa aparecen descritas en el Anexo *"3.1 PLACA CHIPKIT MAX 32"*.

Alimentación de las placas

Para la alimentación del nodo maestro se recomienda emplear la proporcionada por el ordenador a través de la conexión USB 2.0.

Los nodos esclavos se deberán alimentar a través de una fuente externa a través del conector de alimentación externo. Aunque la placa soporte tensiones de hasta 20V, el valor de tensión de entrada recomendado está entre 7-15V, ya que por debajo de este voltaje no se asegura que se pueda obtener una corriente por pin de ±18mA.

Conservacion

Aunque el alcance de este proyecto no contempla su implementación en un proceso real, se recomienda resguardar tanto las placas como el circuito de conexión a la red. Por ello se propone su conservación en:

- Caja con protección IP65, estanco al polvo y a chorros de agua en todas sus direcciones
- Cuadro eléctrico

Transceiver CAN MCP2551

Las especificaciones y características del chip MCP2551 aparecen descritas en el Anexo *"3.3 MCP2551"*. No obstante, las recomendaciones para su correcto funcionamiento en el sistema son las siguientes:

- Tensión de alimentación: 7V máx
- Rango de tensión aceptada en los pines TxD, RxD, Vref, Vs: -0.3V – Vdd+0.3V
- Rango de tensión aceptada en líneas CAN H, CAN L: ±42V
- Rango de temperatura en funcionamiento: -42ºC - 152ºC

- Rango de temperatura en reposo: -55ºC – 150ºC
- Temperatura máxima en la soldadura de los pines: 10 segundos a 300ºC
- Aislamiento de protección de los pines CAN H, CAN L: 6kV
- Aislamiento de protección del resto de pines: 4kV

Cables

Conexión Modo Maestro - PC

La conexión de la placa con el PC se realiza a través de un cable USB 2.0 de tipo A a tipo B mini de 5 pines. La distancia entre el PC y la placa no debe ser mayor a 5 metros.

Conexión de la red de nodos

Para su implementación en un sistema real, la transmisión de señales para la comunicación entre nodos se realizará a través de un cable UTP, un par trenzado sin apantallar. Si se desea una mayor protección frente a interferencias electromagnéticas externas se puede emplear un cable apantallado. La impedancia característica del cable debe ser de 120Ω.

El estándar CAN a diferencia del USB no especifica ningún tipo de conector para el bus, y por tanto cada aplicación puede tener un conector distinto. En cuanto a la longitud del cable, ésta viene limitada por la tasa de transferencia. Para una tasa de 1Mb/s, la longitud se limita a 40 metros, mientras que para tasas de 100 y 200 Kb/s se permiten longitudes de 100 y 200 metros respectivamente.

Software

Se debe disponer de un PC con sistema operativo recomendado de Windows 7 o superior. Este PC deberá tener unos requisitos mínimos:

- Procesador de 1,6 GHz o superior
- 1GB de memoria RAM
- El espacio del disco duro mínimo dependerá de las necesidades de tamaño del buffer Excel.

5.4 CONDICIONES DE EJECUCIÓN Y MONTAJE

5.4.1 Condiciones de configuración

A la hora de configurar el sistema, es necesario seguir unas pautas:

Cada entrada de cada nodo únicamente permite la configuración de 10 lecturas al día. Para evitar la concentración de lecturas en breves periodos de tiempo y aumentar la carga del trabajo del sistema, es recomendable no configurar demasiadas lecturas a la misma hora. Aunque el sistema está preparado para ello, es más aconsejable desfasar unos minutos algunas muestras.

Si por algún motivo alguno de los nodos fallase, no es necesario cambiar la configuración del sistema. Únicamente es necesario sustituir el nodo averiado por otro funcional que no esté operativo en ese momento, y asignarle externamente la misma dirección en la red.

5.4.2 Condiciones de puesta en marcha y mantenimiento preventivo

La instalación de la aplicación y el software necesario, así como la formación a los usuarios será llevada a cabo por el diseñador del proyecto. También será el encargado de realizar el mantenimiento y las reparaciones en caso de fallo o avería. Se deberá revisar con frecuencia el estado de los cables de conexión, así como el correcto funcionamiento de las placas y limpiezas a fondo de las mismas.

Es recomendable realizar periódicamente una revisión de la base de datos, borrando bases y tablas inutilizadas para mejorar la fluidez de la misma.

Se proporcionará dos años de garantía cubriendo los fallos derivados del funcionamiento normal de éste, no cubriendo daños producidos por una mala utilización.

5.4.3 Errores del proyecto

Ante cualquier fallo o error tanto de diseño como de funcionamiento se informará al proyectista para poder dar solución al problema.

En ese caso, se recomienda la desconexión inmediata del sistema para evitar posibles daños tanto de los propios componentes como externos, hasta que el problema se vea resuelto.

5.5 DISPOSICIÓN FINAL

Las dos partes contratantes, tanto dirección técnica como el cliente, se ratifican en el contenido del siguiente Pliego de Condiciones, el cual tiene igual validez, a todos los efectos, que una escritura pública, prometiendo su fiel cumplimiento.

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

6. MEDICIONES

Daniel Pérez García

Septiembre 2018

ÍNDICE DE MEDICIONES

6.1 ESTADO DE MEDICIONES

Unidad de proyecto 1: Componentes electrónicos		
Código	**Descripción**	**Cantidad**
CD001	Placa chipKit MAX 32	11 u
CD002	Fuente de alimentación 15V	10 u
CD003	Placa de Cobre para PCB, FR4, grosor 35µm, 100 x 160 x 1.6mm	11 u
CD004	Resistencia 10K	50 u
CD005	Interruptor DIP, 4 vías	11 u
CD006	Potenciómetro	11 u
CD007	Tiras pin macho macho, separación 2,54 mm	20 u
CD008	Regleta de terminales PCB, 2 tomas	11 u
CD009	MCP2551	11 u
CD010	Taladrado de PCB	11 u
CD011	Grabado y barnizado de pistas	11 u
CD012	Soldado de componentes	11 u
CD013	Caja para montajes electrónicos IP65	11 u
CD014	Sonda de temperatura DS18B20	10 u
CD015	Electrodo industrial PH (SEN0169) compatible con chipKit	10 u

Unidad de proyecto 2: Equipos informáticos		
Código	**Descripción**	**Cantidad**
CD0016	PC portátil	1 u

Unidad de proyecto 3: Cables y conectores		
Código	**Descripción**	**Cantidad**
CD017	Bobina par trenzado (100m)	1 u
CD018	Cable USB, 1m, Mini USB macho tipo A a Mini USB macho tipo B	1 u

Unidad de proyecto 4: Mano de obra		
Código	**Descripción**	**Cantidad**
CD019	Programación de firmware de las placas	35 hr
CD020	Diseño de PCB	30 hr
CD021	Programación del software	70 hr
CD022	Montaje e instalación	60 hr

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un Data Logger mediante CAN bus, aplicado al proceso de fermentación del vino

7. PRESUPUESTO

Daniel Pérez García

Septiembre 2018

ÍNDICE DEL PRESUPUESTO

Este presupuesto se ha desarrollado para un prototipo para un sistema de 10 nodos. No contempla descuentos por volúmenes de compra

7.1 CUADRO DE PRECIOS UNITARIOS

Precios unitarios de mano de obra y elementos auxiliares		
Código	**Descripción**	**Precio**
CD001	Placa chipKit MAX 32	42,70 €
CD002	Fuente de alimentación 15V	19,34 €
CD003	Placa de Cobre para PCB, FR4, grosor 35µm, 100 x 160 x 1.6mm	3,64 €
CD004	Resitencia 10K	0,13 €
CD005	Interruptor DIP, 4 vías	1,17 €
CD006	Potenciómetro	0,61 €
CD007	Tiras pin macho macho, separación 2,54 mm	0,49 €
CD008	Regleta de terminales PCB, 2 tomas	0,57 €
CD009	MCP2551	1,02 €
CD010	Taladrado de PCB	9,85 €
CD011	Grabado y barnizado de pistas	16,00 €
CD012	Soldado de componentes	6,00 €
CD013	Caja para montajes electrónicos IP65	5,65 €
CD014	Sonda de temperatura DS18B20	7,68 €
CD015	Electrodo industrial PH (SEN0169) compatible con chipKit	59,90 €
CD016	PC portátil Intel Celeron /4GB/15.6"	258,59 €
CD017	Bobina par trenzado (100m)	123,21 €
CD018	Cable USB, 1m, Mini USB macho tipo A a Mini USB macho tipo B	5,21 €
CD019	Programación de firmware de las placas	12,00 €
CD020	Diseño de PCB	12,00 €
CD021	Programación del software	12,00 €
CD022	Montaje e instalación	12,00 €

7.2 PRESUPUESTOS PARCIALES

Unidad de proyecto 1: Componentes electrónicos				
Código	**Descripción**	**Cantidad**	**Precio**	**Importe**
CD001	Placa chipKit MAX 32	11 u	42,70 €	469,70 €
CD002	Fuente de alimentación 15V	10 u	19,34 €	193,40 €
CD003	Placa de Cobre para PCB, FR4, grosor 35µm, 100 x 160 x 1.6mm	11 u	3,64 €	40,04 €
CD004	Resistencia 10K	50 u	0,13 €	6,50 €
CD005	Interruptor DIP, 4 vías	11 u	1,17 €	12,87 €
CD006	Potenciómetro	11 u	0,61 €	6,71 €
CD007	Tiras pin macho macho, separación 2,54 mm	20 u	0,49 €	9,80 €
CD008	Regleta de terminales PCB, 2 tomas	11 u	0,57 €	6,27 €
CD009	MCP2551	11 u	1,02 €	11,22 €
CD010	Taladrado de PCB	11 u	9,85 €	108,35 €
CD011	Grabado y barnizado de pistas	11 u	16,00 €	176,00 €
CD012	Soldado de componentes	11 u	6,00 €	66,00 €
CD013	Caja para montajes electrónicos IP65	11 u	5,65 €	62,15 €
CD014	Sonda de temperatura DS18B20	10 u	7,68 €	76,80 €
CD015	Electrodo industrial PH (SEN0169) compatible con chipKit	10 u	59,90 €	599,00 €

Total unidad de proyecto 1: Componentes Electrónicos… 1.844,81€

Asciende la citada partida de "Componentes electrónicos" a la cantidad de **mil ochocientos cuarenta y cuatro euros con ochenta y un céntimos**.

Unidad de proyecto 2: Equipos informáticos				
Código	**Descripción**	**Cantidad**	**Precio**	**Importe**
CD0016	PC portátil Intel Celeron /4GB/15.6"	1u	258,59 €	258,59 €

Total unidad de proyecto 2: Equipos informáticos… 258,59€

Asciende la citada partida de "Equipos informáticos" a la cantidad **doscientos cincuenta y ocho euros con cincuenta y nueve céntimos.**

Unidad de proyecto 3: Cables y conectores				
Código	**Descripción**	**Cantidad**	**Precio**	**Importe**
CD017	Bobina par trenzado (100m)	1 u	123,21 €	123,21 €
CD018	Cable USB, 1m, Mini USB macho tipo A a Mini USB macho tipo B	1 u	5,21 €	5,21 €

Total unidad de proyecto 3: Cables y conectores…128,42€

Asciende la citada partida de "Cables y conectores" a la cantidad de **ciento veintiocho euros con cuarenta y dos céntimos**.

Unidad de proyecto 4: Mano de obra				
Código	**Descripción**	**Cantidad**	**Precio**	**Importe**
CD019	Programación de firmware de las placas	35 hr	24,00 €	840,00 €
CD020	Diseño de PCB	30 hr	20,00 €	600,00 €
CD021	Programación del software	70 hr	24,00 €	1.680,00 €
CD022	Montaje e instalación	60 hr	15,00 €	900,00 €

Total unidad de proyecto 4: Mano de obra…4020,00€

Asciende la citada partida de "Mano de obra" a la cantidad de **cuatro mil veinte euros**.

7.3 PRESUPUESTO GLOBAL

Unidades de proyecto	**Precio**	**%**
Componentes electrónicos	1.844,81 €	*29,51%*
Equipos informáticos	258,59 €	*4,14%*
Cables y conectores	128,42 €	*2,05%*
Mano de obra	4.020,00	*64,30%*

TOTAL EJECUCIÓN MATERIAL…	6.251,82 €
13% Gastos generales…	812,74 €
6% Beneficio industrial…	375,11 €
SUMA DE GG + BI…	1.187,85 €
21% I.V.A…	1.437,29 €
TOTAL PRESUPUESTO GENERAL…	**8.876,96 €**

El presupuesto general asciende a la cantidad de OCHO MIL OCHOCIENTOS SETENTA Y SEIS EUROS CON NOVENTA Y SEIS CÉNTIMOS.

El citado presupuesto ha utilizado precios vigentes de agosto del 2018. Si transcurriera más de 12 meses hasta su realización, los precios deberán ser actualizados a los precios vigentes de ese momento.